UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Modelización metabólica de comunidades microbianas estables crecidas con fuentes de carbono y energía únicas y simples

Máster Universitario en Bioinformática y Biología Computacional

Autor: TALAVERA MARCOS, Silvia

Tutor: AGUIRRE DE CÁRCER GARCÍA, Daniel

Departamento de Biología, Facultad de Ciencias

FECHA: Febrero/2021

AGRADECIMIENTOS

Este trabajo no habría sido posible sin el uso del nodo UAM-BIO del Centro de Computación Científica de la Universidad Autónoma de Madrid, CCC-UAM. Gracias también al Dr. Daniel Machado por sus aclaraciones constantes sobre las herramientas usadas.

A mi tutor, por guiarme no solo en este trabajo sino también en mi incipiente carrera. A todos los profesores que me han ayudado a decidir qué camino tomar en mi carrera y han fomentado mis ganas de aprender.

Le doy las gracias también a mi pareja por su apoyo y su paciencia. Y, sobre todo, gracias a mi familia por apoyarme en todas mis decisiones.

Índice

Acrónimos

ECG	*ecologically-cohesive group*	grupo ecológicamente coherente
PCG	*phylogenetic core group*	grupo filogenético central
CBM	*constraint-based models/modelling*	modelos/modelado basado en restricciones
FBA	*flux balance analysis*	análisis de balance de flujo
ESV	*exact sequence variant*	variante de secuencia exacta
MRO	*metabolic resource overlap*	superposición de recursos metabólicos
MIP	*metabolic interaction potential*	potencial de interacción metabólica
SBML-FBC2	*systems biology markup language - flux balance constraints*	lenguaje de marcado para biología de sistemas - restricciones de balance de flujo

1. INTRODUCCIÓN

En la naturaleza los microorganismos se presentan generalmente en forma de comunidades microbianas, grupos de poblaciones que coexisten en el espacio y el tiempo y son capaces de interaccionar (Nemergut *et al.*, 2013). Cada población está a su vez compuesta por individuos genéticamente homogéneos cuyo material genético define sus capacidades metabólicas y por tanto, hasta cierto punto, el papel y la distribución que puede tener en su comunidad (Ho, Di Lonardo y Bodelier, 2017). Asimismo, la conformación de una comunidad u otra depende, entre otras cosas, de los nichos definidos por el hábitat en cuestión; por ejemplo, se ha comprobado que la cantidad de nutrientes y energía del medio define qué tipo de interacción hay entre dos o más especies (Hoek *et al.*, 2016).

Elaborar una teoría que sea capaz de explicar cómo se ensamblan las comunidades microbianas y, como consecuencia, pueda predecir sus respuestas a cambios en el entorno es un gran reto para la ecología microbiana, con importantes aplicaciones tecnológicas en agricultura (Dessaux, Grandclément y Faure, 2016), industria y salud (Zmora *et al.*, 2016). Un ejemplo son los trasplantes de microbiota fecal para tratar enfermedades relacionadas con la disbiosis de la microbiota intestinal. La posibilidad de utilizar comunidades microbianas cultivadas *in vitro* pero con las mismas propiedades terapéuticas que la materia fecal solventaría varios inconvenientes actuales de esta técnica, como son el gasto económico (Craven *et al.*, 2017) o el estigma social (Anderson, Edney y Whelan, 2012), y permitiría alcanzar un mayor grado de compatibilidad con el paciente (Li *et al.*, 2016). Otros casos en los que el conocimiento del ensamblaje de comunidades microbianas es clave son los tratamientos para disbiosis de otros tipos, tanto en humanos (oral, vaginal, cutánea…) como en animales y plantas. En el caso de las plantas, conocer cómo desarrollar y mantener comunidades microbianas como las que se ha observado que protegen de enfermedades (Mendes *et al.*, 2011), sequía o salinidad podría servir para crear cubiertas

microbianas para semillas (Rocha *et al.*, 2019) que mejoraran la eficiencia y los valores nutricionales de cultivos agrarios, además de reducir el uso de agroquímicos.

Entre los esfuerzos teóricos para explicar el ensamblaje de comunidades microbianas, se incluye el marco teórico desarrollado por Aguirre de Cárcer (2019). El presente Trabajo de Fin de Máster pone el foco en dicho marco teórico, que se introduce brevemente a continuación.

Dado que el ensamblaje de la comunidad se da a nivel de función, es posible que comunidades funcionalmente equivalentes tengan especies diferentes, siempre y cuando realicen las mismas funciones (Parras-Moltó y Aguirre de Cárcer, 2020). Esto implica, en teoría, que pueda haber poblaciones ecológicamente redundantes para un determinado ecosistema; son los llamados grupos ecológicamente coherentes, o ECGs por sus siglas en inglés (Aguirre de Cárcer, 2019), cuyos integrantes pueden estar estrechamente relacionados desde el punto de vista filogenético o no.

Se ha visto tanto en laboratorio como computacionalmente que hay una tendencia, no debida al azar, a la existencia de poblaciones estrechamente emparentadas presentes en todas las instancias del mismo ecosistema microbiano, fenómeno comúnmente denominado señal filogenética. Un ejemplo de esta observación son los resultados de Goldford *et al.* (2018): múltiples réplicas de cultivo en medios sintéticos con distintos inóculos muestran una gran variabilidad a nivel poblacional, pero convergen a nivel de familia en cada medio distinto. Es el caso de los cultivos en medio mínimo M9 con glucosa o con citrato, en los que aparecen decenas de especies asociadas a únicamente dos familias: Pseudomonadaceae y Enterobacteriaceae.

Para justificar estas observaciones, el marco teórico propuesto establece que las poblaciones estrechamente emparentadas presentes siempre en cada réplica de un ecosistema representan especies funcionalmente cohesivas debido a su cercanía filogenética y, por tanto, afectadas por las mismas fuerzas selectivas. Estas poblaciones se definen como PCGs (grupos filogenéticos centrales). En realidad, estos grupos son también ECGs: se trata de poblaciones que presentan un conjunto de características, en este caso conservadas filogenéticamente, que les permiten superar la selección por parte de un conjunto de factores bióticos y abióticos. Se trata pues de grupos muy cohesivos ecológicamente y por tanto con una estructura interna plausiblemente regida por competición interna y por procesos neutrales como deriva e inmigración. Este marco teórico reconcilia en parte el neutralismo y las explicaciones basadas en nichos, dos visiones frecuentemente opuestas (Wennekes, Rosindell y Etienne, 2012).

El planteamiento derivado de esta idea es que las comunidades microbianas siguen un diseño modular: cada ECG constituye una partición o módulo en sí mismo y una comunidad viene dada por la combinación de múltiples módulos necesarios para la misma. Cada módulo desempeñaría una función en el ecosistema, función que, teniendo en cuenta la naturaleza del mundo microbiano, probablemente esté muy relacionada con el metabolismo. La existencia de diferentes módulos en una comunidad podría estar en buena medida determinada por interacciones metabólicas impuestas por los condicionantes abióticos del sistema. Así, cada instancia de un mismo ecosistema presentaría al menos una población de cada ECG, con sus correspondientes roles ecológicos similares. Se ha demostrado recientemente que esta descripción modular es adecuada para ciertas comunidades reales (Enke *et al.*, 2019; Estrela *et al.*, 2020).

1.1. Modelado matemático de comunidades microbianas

Los ecosistemas microbianos son sistemas muy complejos, de manera que para comprender los fenómenos subyacentes a su ensamblaje requerimos la capacidad de abstracción de los modelos matemáticos. Podemos hablar de dos clases principales de modelos matemáticos deterministas aplicados al modelado de comunidades microbianas: modelos de ecuaciones diferenciales

(modelos dinámicos) y modelos estequiométricos basados en restricciones (modelos metabólicos) (Succurro y Ebenhöh, 2018). Estos últimos son los que presentan un mayor potencial a la hora de explicar el ensamblaje de la comunidad (Aguirre de Cárcer, 2020) y son los que se emplean en este trabajo.

Es posible crear modelos con un gran poder predictivo teniendo en cuenta solamente series temporales de composición de una comunidad. Un ejemplo clásico es el modelo de Lotka-Volterra, que tiene en cuenta una sencilla relación depredador-presa. A partir de él se han creado modelos Lotka-Volterra generalizados (gLV) que permiten la presencia de múltiples poblaciones coexistentes (aunque estudiadas por parejas) y se ha demostrado que pueden describir adecuadamente ciertos ecosistemas (Stein *et al.*, 2013). Los modelos gLV son sistemas de ecuaciones diferenciales ordinarias cuyos parámetros comprenden, para cada población, la tasa de crecimiento y una serie de descriptores de la fuerza de su interacción con cada una de las demás poblaciones del sistema. En ellos se asume un medio abiótico constante (Gonze *et al.*, 2018).

La limitación de los gLV yace en el hecho de que las interacciones en el ecosistema se limitan al estudio de interacciones por parejas. No está claro si el estudio de interacciones por parejas es capaz de reflejar el comportamiento real de una comunidad (Momeni, Xie y Shou, 2017), especialmente al no tener en cuenta los factores abióticos. Además, estos modelos no proporcionan información sobre las causas subyacentes a las dinámicas observadas.

Una alternativa a los modelos de ecuaciones dinámicas son los modelos mecanísticos, que se basan en las interacciones mediadas por metabolitos. En ellos, las dinámicas de la comunidad se simulan en base al conocimiento previo de las interacciones de los individuos con un *pool* común de metabolitos. Estos modelos parametrizan cada especie en base a las interacciones entre los metabolitos de sus reacciones, de manera que cada especie queda definida como una matriz de reacciones dada por cada compuesto implicado en la misma y su correspondiente coeficiente estequiométrico. Se ha comprobado que los modelos mecanísticos funcionan mejor que los modelos de Lotka-Volterra generalizados (Momeni, Xie y Shou, 2017; Brunner y Chia, 2019) aunque requieren de información *a priori*.

1.1.1. Modelado metabólico: modelos basados en restricciones

Las interacciones microscópicas dentro de una comunidad microbiana determinan la distribución y formación de agregados celulares a escala microscópica: los organismos unicelulares se agregan si interaccionar les beneficia y se segregan si antagonizan. Estas agregaciones a pequeña escala (~100 μm) se han observado en comunidades de todo tipo, desde terrestres y acuáticas hasta industriales, pasando por animales, y afectan a propiedades macroscópicas de la comunidad como la eficiencia del uso de recursos o las tasas de producción de biomasa (Cordero y Datta, 2016). Estudiar las interacciones metabólicas a este nivel es imprescindible para comprender el ensamblaje microbiano *(Figura 1*), y el modelado metabólico nos permite hacer esto computacionalmente.

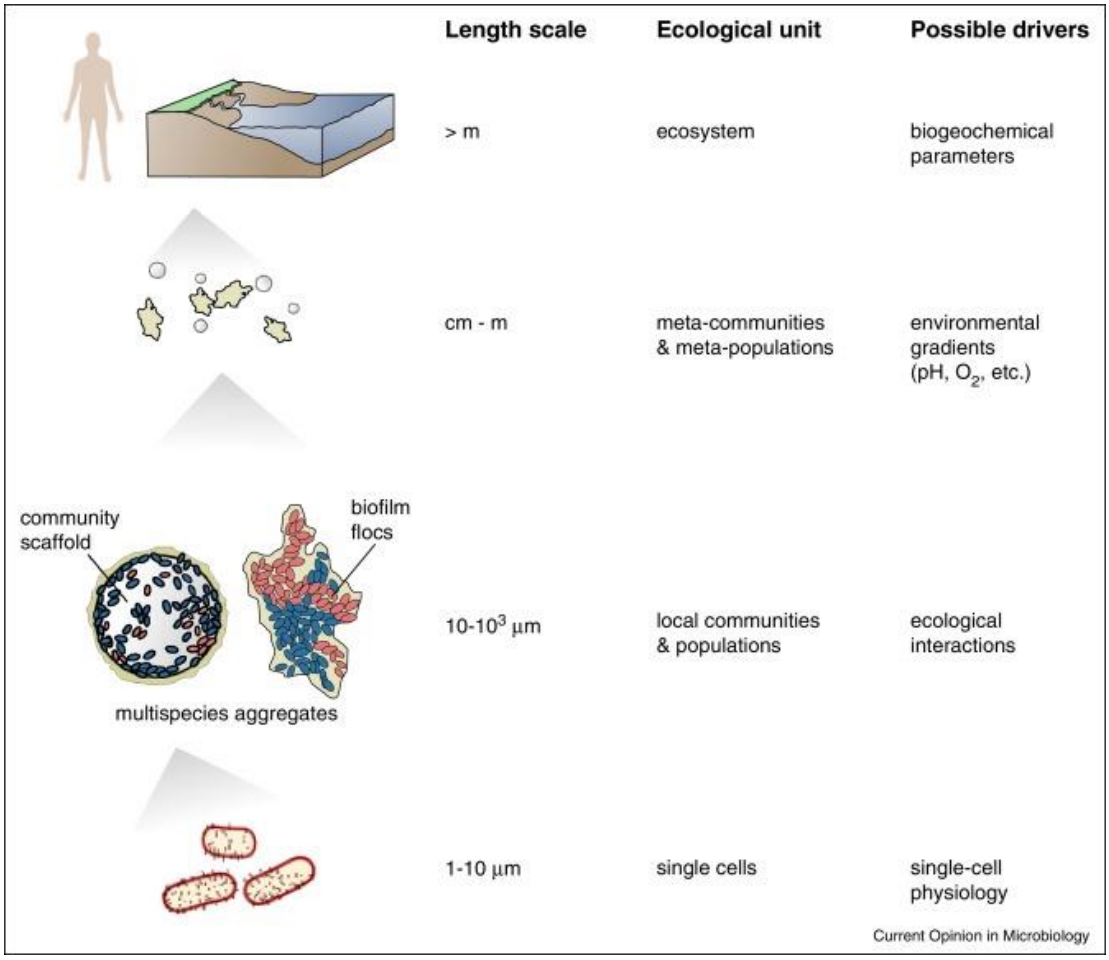

Figura 1. ***Diferentes escalas de los ecosistemas microbianos***. La menor escala es la celular. La mayor escala, de metros o kilómetros, muestra una gran estabilidad ambiental aparente. En medio tenemos, por un lado, las meta-comunidades que se pueden estudiar a escala de menos de un metro o pocos metros y, por otro, el nivel más relevante a la hora de estudiar ensamblaje de comunidades microbianas: los agregados celulares que componen comunidades individuales. A este nivel, las interacciones biológicas tienen un efecto medible en las dinámicas y la composición poblacional. Adaptado de Momeni, Xie y Shou (2017).

Los modelos basados en restricciones o CBM (*constraint-based models*) representan cada red de reacciones bioquímicas como una matriz estequiométrica *S* de dimensiones *(m,r)*, siendo *m* los metabolitos existentes en esa red y *r* las reacciones posibles. Cada elemento M_{ij} es el coeficiente estequiométrico del metabolito *i* en esa reacción *j* (*Figura 2*). Un coeficiente negativo indica que el metabolito se está consumiendo y uno positivo, que se está produciendo. Si es cero, ese metabolito no participa en la reacción; se trata por tanto de una matriz escasa, o *"sparse"* (Orth, Thiele y ØPalsson, 2010). El equilibrio de masas de los metabolitos da lugar a sistemas de ecuaciones diferenciales, que tienen la forma del producto de la matriz *S* y el vector de flujos de reacción *v*. El vector *v* tiene una longitud *r* y, como se trata de un caso en el que hay más reacciones *r* que metabolitos *m*, hay múltiples soluciones a la ecuación $Sv = 0$.

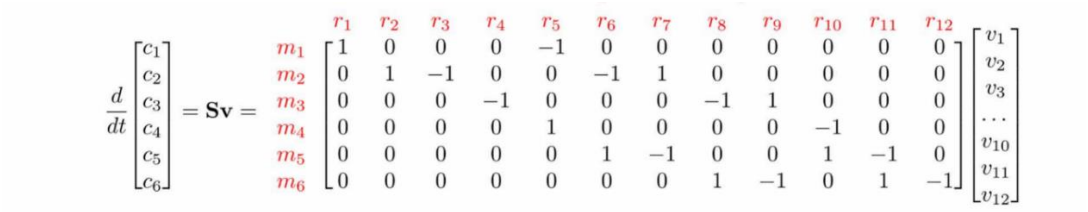

Figura 2. Los modelos basados en restricciones o CBM representan las redes metabólicas matemáticamente como matrices estequiométricas (S). Cada reacción *r* tiene un coeficiente estequiométrico para cada metabolito *m*. La matriz *S* se multiplica por un vector de flujos de reacción (*v*) para obtener una serie de coeficientes c_n. Adaptado de Succurro y Ebenhöh (2018).

La resolución de estos sistemas de ecuaciones parte de 1) la asunción de estado estacionario y 2) el establecimiento de restricciones al flujo de varias reacciones. Estas restricciones se basan en consideraciones termodinámicas y también en información experimental. Por ejemplo, las restricciones en las reacciones de importación de nutrientes dependen de su disponibilidad.

En el caso del análisis de modos elementales o EMA (*elementary mode analysis*), la resolución de las ecuaciones derivadas de la matriz *S* tiene como objetivo identificar las rutas metabólicas mínimas ("modos elementales") para que se dé un estado estacionario en la red. Es decir, las distribuciones de flujo de todos los estados estacionarios posibles en el sistema pueden ser descritas como una combinación lineal de estos modos elementales (Schuster y Hilgetag, 1994). En principio, este tipo de resolución no requiere información previa (aparte de las restricciones termodinámicas), pero se pueden añadir al modelo posteriormente condiciones ambientales para curar el modelo. Se trata de un proceso computacionalmente exigente.

Por otro lado, el análisis de balance de flujo o FBA (*flux balance analysis*) es un método más rápido cuyo objetivo es obtener una única distribución de flujos. Este tipo de resolución se centra en un único problema (es decir, una única reacción) que hay que optimizar, lo cual es posible mediante programación lineal. La función a optimizar suele ser la reacción de producción de biomasa o tasa de crecimiento (Sahraeian y Yoon, 2013).

La solución devuelta por el FBA viene determinada por la función a optimizar elegida. Además, puede dar lugar bien a una única solución óptima (que describe un único estado estacionario) o bien a varias subóptimas, de las cuales se selecciona una prácticamente al azar (Machado, comunicación personal). Sin embargo, su gran ventaja es la velocidad, que permite integrar datos masivos tales como los meta-ómicos.

De entre las variantes de FBA existentes cabe resaltar el análisis de balance de flujo con asignación restringida o CAFBA (*constrained allocation flux balance analysis*), que refina el cálculo incluyendo los costes de biosíntesis de proteínas, incluso proteínas desconocidas. Para ello se incluyen tres parámetros globales basados en datos comprobados experimentalmente (Mori *et al.*, 2016).

La mayor limitación de los métodos CBM es que se puede perder información sobre el dinamismo del sistema que no se perdería con modelos de ecuaciones diferenciales dinámicas. Por ejemplo, no es posible predecir las concentraciones finales de los metabolitos. En la actualidad se están desarrollando métodos que intentan integrar ambos enfoques (Bauer *et al.*, 2017).

Al igual que para los gLV, los CBM requieren conocimiento previo. En este caso, en vez de emplear series temporales de composición de la comunidad, se necesita tener una idea de las reacciones metabólicas que puede tener una determinada especie. Esta información se obtiene a partir de estudios *in vitro* o datos genómicos. De hecho, se pueden generar modelos metabólicos directamente desde bases de datos de anotaciones genómicas, que nos aportan las funciones conocidas, y a continuación se modelan las interacciones computacionalmente (Henry *et al.*, 2010).

1.2. Hipótesis de trabajo y objetivos

El objetivo general de este proyecto es contrastar la hipótesis arriba mencionada que predice cohesión ecológica intra-grupo y modularidad de los PCGs en la construcción de la comunidad. Para ello, los objetivos específicos contemplados son:

A) Desarrollar herramientas en forma de *scripts* que permitan el análisis automatizado de las interacciones metabólicas entre miembros de PCGs de distintos ecosistemas microbianos.
B) Analizar un caso real partiendo de datos de comunidades cultivadas en medios con una única fuente de energía y carbono.

2. METODOLOGÍA

A continuación se explican los datos, programas y métodos que se han utilizado en este trabajo. Se incluye una representación esquemática del procedimiento seguido en la *Figura 3*, así como información adicional específica de los scripts desarrollados en el Anexo.

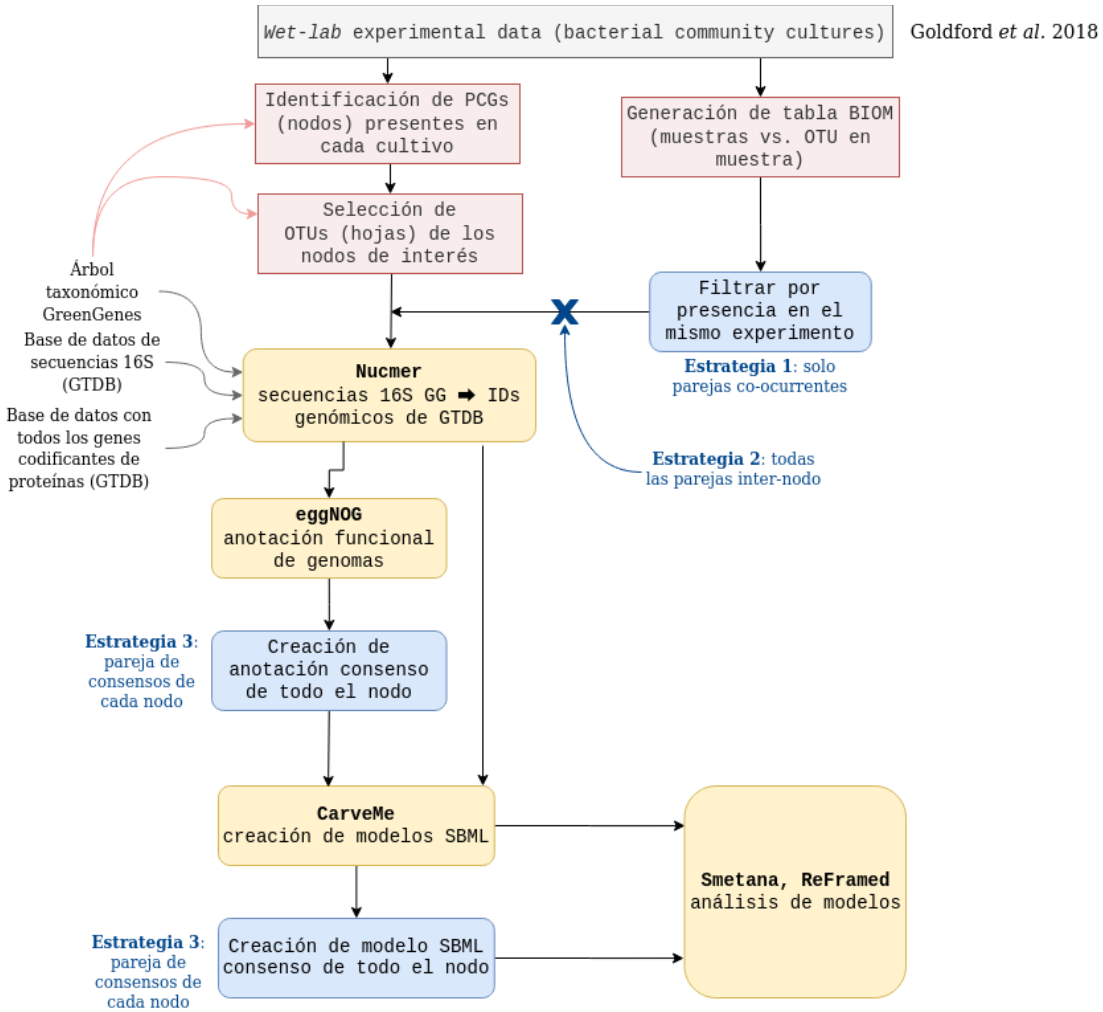

Figura 3. Representación esquemática del plan de trabajo. Cuadros rojos: pasos previos. Cuadros amarillos: herramientas principales utilizadas. Cuadros azules: pasos de procesado de datos.

2.1. Datos experimentales

Goldford *et al.* (2018) cultivaron comunidades microbianas en tres variantes del medio mínimo M9, cada una suplementada con una única fuente de energía y carbono: glucosa, leucina o citrato (*Figura 4*). En dichos medios se cultivaron 12 inóculos diferentes extraídos de ecosistemas de suelo y hojas, y, tras su estabilización mediada por 12 pases (aproximadamente 84 generaciones), se hizo secuenciación masiva de los genes marcadores filogenéticos ARNr 16S (secuenciación pareada con Illumina MiSeq). Se hicieron 8 réplicas para cada medio de cultivo. Los autores observaron que, a nivel de ESV (*exact sequence variant*, variante de secuencia exacta), el nivel menos profundo de filogenia detectable mediante este acercamiento, había una gran variabilidad entre las réplicas de la misma fuente de carbono, pero no la había apenas al nivel taxonómico de familia. Como ya se mencionó en la introducción, estas familias son un ejemplo de los PCGs predichos en el marco teórico estudiado (Aguirre de Cárcer, 2019).

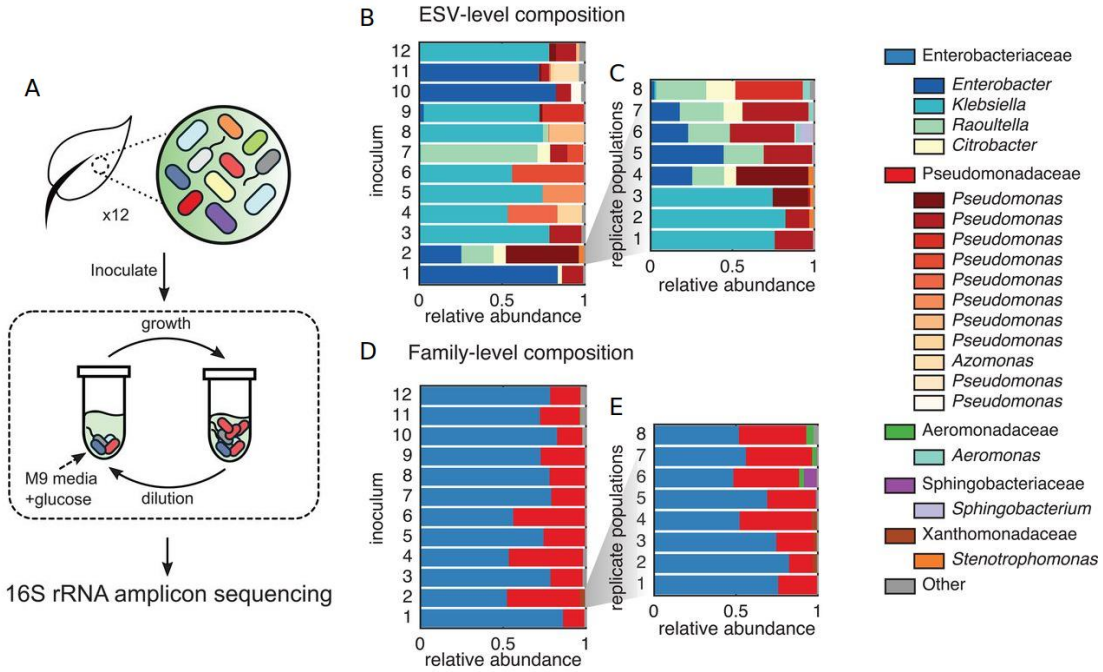

Figura 4. Diseño y resumen de los resultados de los experimentos de Goldford et al. (2018). A) Diseño experimental: 12 muestras de comunidades microbianas de suelo y hojas se inocularon en cultivos, y luego fueron sometidas a ciclos de cultivo y dilución. Finalmente, se determinó la composición bacteriana de la comunidad en cada pase mediante secuenciación de amplicones de ARNr 16S. B) Composición bacteriana a nivel de ESV (variante de secuencia exacta). C) Distribución de la composición a nivel de ESV en 8 inóculos. Hay una gran variabilidad a este nivel. D) Composición a nivel taxonómico de familia. E) Distribución de la composición a nivel de familia en 8 inóculos. La variabilidad es pequeña en este caso, convergiendo en todos los casos en dos familias destacadas, Pseudomonadaceae y Enterobacteriaceae. Adaptado de Goldford *et al.* (2018).

2.2. Herramientas utilizadas

2.2.1. *BacterialCore.py*

Se ha utilizado una herramienta específica, previamente desarrollada por el equipo de investigación en el cual se ha realizado este Trabajo de Fin de Máster (Parras-Moltó y Aguirre de Cárcer, 2020) para identificar PCGs [*BacterialCore.py]* en datos procedentes de secuenciación masiva del ARNr 16S.

El árbol taxonómico de referencia utilizado fue Greengenes v. 13.5. Se trata de un árbol que utiliza términos taxonómicos relativamente antiguos (2013), pero disponible públicamente (https://greengenes.secondgenome.com/?prefix=downloads/greengenes_database/gg_13_5).

El presente análisis de PCGs parte de las secuencias 16S correspondientes, y también de las secuencias 16S de referencia de Greengenes *gg_13_5*. Los nodos del árbol se etiquetaron con la función de R *makeNodeLabel* del paquete *ape*.

2.2.2. NUCmer

NUCmer es un alineador de secuencias de nucleótidos (NUCleotide MUMmer) perteneciente al paquete MUMmer (Kurtz *et al.*, 2004). En este trabajo se ha utilizado NUCmer (versión 3.1) para alinear las secuencias de los genes 16S de los organismos correspondientes a cada nodo de interés con una base de datos de secuencias 16S, la versión 95 de secuencias 16S bacterianas GTDB publicada el 17 de julio de 2020 (*bac120_ssu_reps_r95*, https://data.ace.uq.edu.au/public/gtdb/data/releases/release95/95.0/).

El objetivo de este alineamiento es obtener los números de acceso de sus correspondientes genomas representativos de GTDB. Concretamente, asignamos a cada organismo un archivo FASTA que contiene todas las secuencias codificantes de proteínas en su genoma. A partir de este archivo se creó un modelo metabólico con CarveMe posteriormente (ver abajo). Los archivos FASTA de secuencias codificantes utilizados se encuentran en la base de datos GTDB (*gtdb_proteins_aa_reps_r95*, https://data.ace.uq.edu.au/public/gtdb/data/releases/release95/95.0/genomic_files_reps/).

Se escogió NUCmer por su sencillez y por ser adecuado para esta tarea. NUCmer está diseñado para alinear secuencias altamente conservadas con secuencias de nucleótidos con las que están muy emparentadas, como es el caso de las 16S. Funciona encontrando primero coincidencias exactas de una longitud corta fija y extendiendo luego el alineamiento en ambos sentidos a partir de cada coincidencia. Significativamente, permite un filtrado de los resultados más sencillo que otras herramientas como Bowtie2 o BLAST.

Se dieron por válidas solamente las asignaciones para las cuales hubiera un 90% de cobertura de la secuencia 16S de referencia (la de GTDB) y un 97% de identidad. Este filtro se hizo con la herramienta de MUMmer *show-coords* y con comandos en Awk (véase el Anexo).

2.2.3. CarveMe

CarveMe es una herramienta de reconstrucción metabólica que obtiene un modelo metabólico en formato SBML a partir de un genoma anotado (Machado *et al.*, 2018). Estos modelos contienen información sobre los metabolitos presentes en el sistema, las reacciones en las que participan y su localización, incluyendo el espacio extracelular. También pueden contener restricciones

predefinidas. En este trabajo se han generado modelos con CarveMe (versión 1.4.0) que han sido posteriormente sometidos a análisis metabólico con otras herramientas.

CarveMe es una herramienta interesante porque propone un enfoque novedoso para evitar generar modelos inexactos. Otras herramientas de reconstrucción, como modelSEED (Henry *et al.*, 2010), Merlin (Dias *et al.*, 2015) o PathwayTools (Karp *et al.*, 2015), ofrecen un enfoque *bottom-up*: se parte de genomas anotados, cada anotación se traduce en una reacción bioquímica extraída de una base de datos como BiGG o KEGG, se construye un modelo a partir de estas reacciones y por último se hace una curación manual del modelo. El problema de este enfoque es que, si falta información en las anotaciones, los modelos obtenidos son poco realistas. Lo que hace CarveMe es el proceso opuesto (*top-to-bottom*): además de los genomas anotados, parte de un modelo universal ya creado y funcional. El proceso consiste en "esculpir" el modelo retirando aquellas reacciones bioquímicas que, basándose en las reacciones descritas en las anotaciones, el programa considera que no están en el organismo dado. Por tanto, el modelo final no solo incluye aquellas reacciones especificadas en las anotaciones, sino también otras que no están anotadas pero sí es probable que formen parte del metabolismo del organismo. Este enfoque no solo es novedoso sino también efectivo, ya que, según se revela con la herramienta MEMOTE (Lieven *et al.*, 2020), los modelos de CarveMe tienen una proporción muy baja de reacciones bloqueadas, siendo que una proporción alta es un indicador de baja calidad.

Como hemos mencionado, los modelos generados por CarveMe contienen datos del espacio extracelular. A la hora de crearlos se puede tener en cuenta el medio de crecimiento de la población realizando *gap-filling*, un proceso que completa el modelo con datos clave para asegurar que la población es capaz de crecer en el medio especificado, compensando así las anotaciones que puedan faltar en los datos genómicos. También es posible inicializar posteriormente el modelo en ese medio, es decir, predefinir la composición del medio extracelular para simulaciones posteriores.

2.2.4. Smetana

Smetana (*species metabolic interaction analysis* o "análisis de interacciones metabólicas entre especies") es una herramienta de análisis de modelos metabólicos. Este análisis es aplicado no a una sola especie, sino a dos o más (es decir, a una comunidad) y da información sobre los metabolitos que intercambian, incluyendo qué población es donante y cuál es la receptora, y sobre su interdependencia, mediante una serie de métricas especializadas (Sahraeian y Yoon, 2013). En este trabajo se ha utilizado Smetana 1.2.0.

Smetana tiene en cuenta todos los posibles intercambios y luego realiza un FBA (dependiente de la herramienta ReFramed, del mismo autor (https://github.com/cdanielmachado/reframed)) que optimiza una serie de funciones. La obtención de cada una de las métricas depende de la optimización de un parámetro interno determinado. Esto se consigue con programación lineal en enteros mixta (MILP o *mixed-integer linear programming*). Una descripción más detallada de estas métricas se incluye en la *Tabla 1*. Además, una característica que diferencia Smetana de otras herramientas es que lo único que se asume es que todas las especies puedan sobrevivir con los recursos dados, pero no depende de la asunción de crecimiento óptimo a ningún nivel.

En el modo "global", las métricas computadas son la superposición de recursos metabólicos o MRO por sus siglas en inglés y el potencial de interacción metabólica o MIP. A grandes rasgos, la MRO es una medida de la competición de cada especie con el resto de su comunidad y la MIP es una medida de su cooperación con la misma. Se ilustran estos conceptos en la *Figura 5*.

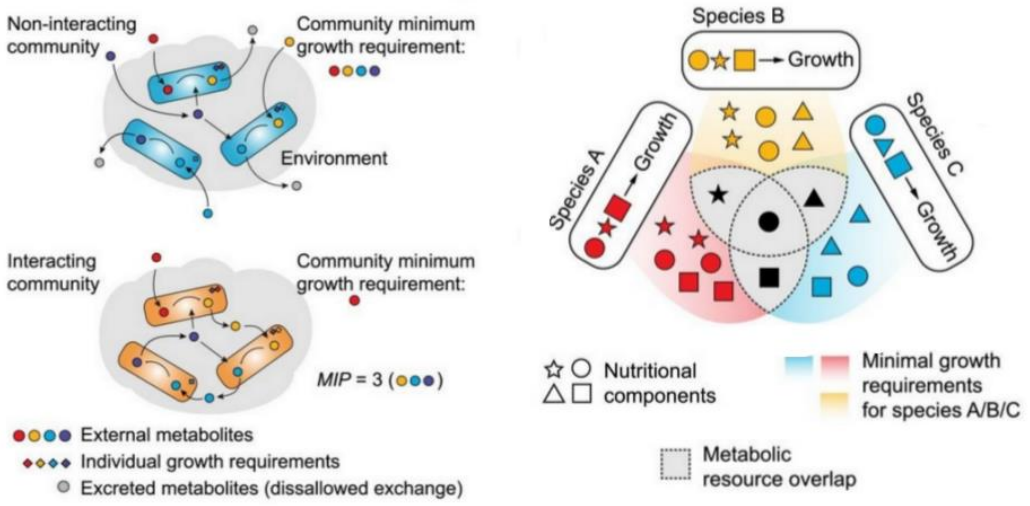

Figura 5. Ilustración de los conceptos de MIP y MRO. Ambas son estimaciones teóricas de la colaboración y la competición en una comunidad. La puntuación MIP consiste en el número de nutrientes que una especie puede intercambiar para disminuir su dependencia de los recursos externos. La MRO representa la superposición de nutrientes que hay entre las distintas especies de la comunidad. Adaptado de Zelezniak et al. (2015).

En el modo "detallado", el objetivo es identificar interacciones obligatorias entre especies en un medio concreto. Por lo tanto, no utiliza un medio completo sino un medio mínimo (que puede ser especificado por el usuario). En estas condiciones, habrá especies que no puedan crecer en solitario en dicho medio pero sí en comunidad, teniendo cierto grado de dependencia. Esta dependencia se describe con la puntuación *smetana*, que a su vez es la suma de varias métricas. La salida que devuelve la ejecución de Smetana en este modo es, además de las métricas, una lista de metabolitos intercambiados entre las diferentes especies. El usuario puede solicitar que se devuelvan solo los intercambios esenciales para el crecimiento o su totalidad.

Tabla 1. Descripción de las métricas definidas por Smetana. Se incluyen el potencial de interacción metabólica o MIP, la superposición de recursos metabólicos o MRO, la puntuación de acoplamiento de especies o SCS, la puntuación de toma de metabolitos so MUS, la puntuación de producción de metabolitos o MPS y la puntuación *smetana*.

Métrica (Abreviatura)	**Descripción**	**Significado**
Metabolic interaction potential ***(MIP)***	Medida de cooperación. Se calcula en el modo global de Smetana. Equivale al máximo número de componentes nutricionales que una comunidad genera para sí misma mediante intercambios, es decir, el número de metabolitos intercambiables para disminuir la dependencia que tienen, como comunidad, de su hábitat. Se calcula comparando su crecimiento en el medio por sí misma con su crecimiento en comunidad.	Cuanto mayor es para una comunidad, mayor potencial de beneficiarse mediante intercambios entre las especies. Si el cálculo falla (*n/a*), quiere decir que al menos una de las poblaciones no es capaz de crecer en el medio indicado por sí misma.
Metabolic resource overlap ***(MRO)***	Medida de la competición. Se calcula en el modo global de Smetana. Cuantifica la similitud entre las necesidades metabólicas de cada población miembro de la comunidad, en el caso de crecer	Cuanto más alta, más riesgo hay de que haya competición entre las poblaciones.

	juntas. Es una puntuación intrínseca de la comunidad e independiente del hábitat.	Si el cálculo falla quiere decir que no pueden crecer como co-cultivo.
Species coupling score ***(SCS)***	Mide la dependencia del crecimiento de una especie A en la presencia de una especie B, siendo ambas miembros de una comunidad de N miembros implicadas en un intercambio. Se calcula en el modo detallado de Smetana. Se puede desactivar este cálculo con la opción *--no-coupling*.	Cada línea del *output* corresponde a un intercambio. Si las especies implicadas son independientes, el cálculo de la SCS falla y el *smetana score* también, eliminando la línea correspondiente a esas dos especies del archivo de salida. Si no hay especies interdependientes, el archivo sale vacío, a menos que se utilice la opción *--no-coupling.*
Metabolite uptake score ***(MUS)***	Medida, en cada intercambio, de cuánto depende el crecimiento de la especie que recibe el metabolito de dicho intercambio. Se calcula en el modo detallado de Smetana.	Cuanto más alta, más importante es el intercambio para la especie que recibe el metabolito.
Metabolite production score ***(MPS)***	Puntuación binaria que indica si el metabolito intercambiado es producido por el donante o no.	Es 0 si no lo produce. Es 1 si sí lo produce.
Smetana score	Medida de la certeza de que haya *cross-feeding* de un metabolito X entre cada pareja de especies. Se calcula en el modo detallado de Smetana. Es el producto de la SCS, la MUS y la MPS y se corresponde con el porcentaje de iteraciones de la simulación en las cuales se da dicho intercambio.	Si falla, significa que las poblaciones no dependen del resto de la comunidad.

2.2.5. eggNOG-mapper

EggNOG-mapper es una herramienta de anotación funcional rápida (https://github.com/eggnogdb/eggnog-mapper/tree/kegg_bigg). Su rapidez se debe a que utiliza la base de datos eggNOG. Esta base de datos consiste en grupos de ortólogos (COGs, *Clusters of Orthologous Groups of proteins*) a los cuales se les ha asignado previamente una serie de funciones (anotaciones), de manera que al ejecutar eggNOG-mapper no hace falta repetir esta asignación. De acuerdo con los autores, la ventaja de utilizar ortólogos en vez de búsquedas por homología es que es menos probable asignar a una secuencia codificante la función de un parálogo que tenga una función distinta (Huerta-Cepas *et al.*, 2017). En este trabajo se ha usado eggNOG-mapper (versión 2.0.1) para la anotación funcional de los archivos de secuencias proteicas GTDB de cada organismo, los cuales se han identificado a su vez gracias al alineamiento de sus secuencias 16S con Nucmer.

2.2.6. ReFramed

ReFramed (https://github.com/cdanielmachado/reframed) es una herramienta en Python para modelado metabólico que emplea diferentes métodos de CBM, incluyendo FBA y CAFBA. Los programas de optimización que utiliza, o *solvers*, son CPLEX o Gurobi. En este trabajo usamos CPLEX.

Esta herramienta es llamada por Smetana para hacer su análisis FBA y por CarveMe en la reconstrucción y el *gap-filling* pero, además, se puede utilizar independientemente. En este trabajo se utilizó ReFramed (versión 1.2.0) para comprobar la tasa de crecimiento en diferentes medios de cada modelo generado por CarveMe.

2.2.7. Scripts desarrollados para este trabajo

Aparte de los programas ya mencionados, durante este trabajo se han desarrollado una serie de *scripts* en R y Python para automatizar el proceso: *modelado.R* incluye el alineamiento con Nucmer, la creación de modelos con CarveMe y (opcionalmente) el análisis con Smetana para una pareja de nodos dada. *annotate.R* se encarga de la anotación funcional con eggNOG-mapper y, opcionalmente, la creación de un modelo consenso. También es capaz de llamar a Nucmer para iniciar el proceso desde el principio. Las funciones definidas para estos *scripts* se encuentran en *utils.R*. Además, se han desarrollado dos *scripts* en Python para crear los consensos: *consenso.py* crea un modelo SBML a partir de otros modelos SBML y *consenso_EGG.py* crea una tabla de anotaciones consenso a partir de múltiples archivos de anotaciones. Por último, *parser.R* y *RefrFBA.py* generan informes sobre los resultados de Smetana y ReFramed. Todos estos *scripts* se encuentran en el Anexo y en https://github.com/urihs/TFM.

2.3. Obtención y examen previo de los PCG

En un trabajo previo al presente, *BacterialCore.py* fue empleado por este grupo de investigación para detectar qué grupos filogenéticos (nodos en el árbol) estaban presentes en todas las muestras de cada uno de los tres medios de cultivo diferentes (*Tabla 2*). Este análisis reveló que los PCG de los medios enriquecidos en citrato y en glucosa eran, para ambos casos, de las familias Enterobacteriaceae (nodo 35562 en ambos casos) y Pseudomonadaceae (nodo 27828 en el caso de la glucosa y 28866 en el caso del citrato). También se encontró un PCG afiliado a Pseudomonadaceae en los datos del medio con leucina (nodo 28853), junto a un nodo más amplio de Proteobacteria (nodo 13821). Además, en el caso del citrato, había presente un tercer nodo muy profundo (94.54% del total de 203452 hojas en el árbol), que se descartó para el análisis.

Tabla 2. Información sobre las PCG detectadas en cada medio de cultivo. Los tres medios eran variantes del medio mínimo M9 con glucosa, leucina o citrato como única fuente de carbono y energía.

Leucina:					
Distancia entre los 16S del nodo					
Nodo	**Número de hojas en experimento**	**Media ± desviación estándar**		**Máxima**	**Número de hojas en nodo y porcentaje del total del árbol**
Nodo13821	22	0.097 ± 0.046		0.162	12054 (5.93%)
Nodo28853	6	0.030 ± 0.011		0.0467	634 (0.31%)
Abundancia y taxonomía					
Nodo	**Mínima**	**Máxima**	**Media**	**Desviación estándar**	**Taxonomía (consenso 90%)**
Nodo13821	0.095	0.752	0.200	0.142	p__Proteobacteria
Nodo28853	0.134	0.966	0.712	0.181	f__Pseudomonadaceae;
Glucosa:					
Distancia entre los 16S del nodo					
Nodo	**Número de hojas en experimento**	**Media Desviación estándar**		**Máxima**	**Número de hojas en nodo y porcentaje del total del árbol**
Nodo35562	7	0.037 ± 0.011		0.052	1390 (0.68%)
Nodo27828	10	0.045 ± 0.018		0.085	1977 (0.97%)

Abundancia y taxonomía					
Nodo	**Mínima**	**Máxima**	**Media**	**Desviación estándar**	**Taxonomía (consenso 90%)**
Nodo35562	0.025	0.982	0.715	0.210	f__Enterobacteriaceae;
Nodo27828	0.010	0.610	0.210	0.141	f__Pseudomonadaceae;
Citrato:					
Distancia entre los 16S del nodo					
Nodo	**Número de hojas en experimento**	**Media ± desviación estándar**		**Máxima**	**Número de hojas en nodo y porcentaje del total del árbol**
Nodo28866	4	0.025 ± 0.013		0.041	423 (0.21%)
Nodo35562	7	0.031 ± 0.010		0.047	1390 (0.68%)
Nodo11102	58	0.182 ± 0.058		0.271	192351 (94.54%)
Abundancia y taxonomía					
Nodo	**Mínima**	**Máxima**	**Media**	**Desviación estándar**	**Taxonomía (consenso 90%)**
Nodo28866	0.004	0.7878	0.5117	0.1413	f__Pseudomonadaceae;
Nodo35562	0.000	0.6582	0.3257	0.1893	f__Enterobacteriaceae;
Nodo11102	0.000	0.9569	0.1626	0.1719	k__Bacteria

2.4. Generación de modelos

Para estudiar computacionalmente las interacciones entre estos PCGs, el primer paso es generar modelos metabólicos. Conociendo los identificadores de Greengenes para las hojas de cada nodo, se usó NUCmer y CarveMe para crear un modelo metabólico de cada organismo. El objetivo era analizar estos modelos por parejas más adelante, perteneciendo cada miembro a uno de los dos PCGs estudiados para cada medio.

En este punto se plantearon tres enfoques diferentes. Se llevaron a cabo los tres, siendo estos análisis paralelos, es decir, aplicándose de igual manera para los tres medios de cultivo pero de forma independiente:

- Generar solamente los modelos de aquellos organismos que estaban presentes en la misma muestra de los experimentos originales de Goldford *et al.* (2018). Se ilustra en la *Figura 3*, *"*Estrategia 1".
- Crear modelos de todos los genomas válidos de cada PCG de interés ("Estrategia 2").
- Crear modelos consenso que representen a cada PCG ("Estrategia 3").

Para seleccionar los organismos presentes en cada experimento, se utilizó la función *check()* definida en *utils.R*. Esta función devuelve solo aquellas parejas de organismos que estén presentes en una misma réplica del medio deseado. La entrada son los datos experimentales en sí, una tabla generada a partir de formato *biom* que contiene los números de hoja Greengenes de cada organismo que crece en cada réplica de ese medio. Es importante mencionar que, en cualquier caso, los organismos seleccionados para generar modelos son solo aquellos que superen el filtro de calidad en el paso de asignación con NUCmer. Se hizo esta selección para los tres medios por separado.

Para acelerar el proceso, sobre todo en el caso de la Estrategia 2, la paralelización fue clave. Por ejemplo, *modelado.R* tarda entre 18 y 20 horas en hacer los alineamientos, generar 34 modelos y analizarlos, mientras que cuando se paraleliza el paso de CarveMe en 256 núcleos el proceso tarda menos de dos horas.

Como ya se ha explicado, CarveMe genera modelos a partir de un modelo universal que es "esculpido". Este modelo universal puede ser genérico o puede ser específico de bacterias Gram-

positivas o Gram-negativas; el modelo genérico no incluye los componentes de la pared celular específicos de estos dos tipos, lo que puede dar lugar a falsos negativos a la hora de predecir qué genes son esenciales en las rutas de biosíntesis de lípidos (Machado *et al.*, 2018). Por eso, el *script modelado.R* selecciona automáticamente el modelo universal adecuado a partir de la taxonomía de los PCG (dada como datos de entrada).

Los modelos generados están en formato XML, concretamente SBML-FCB (*Systems Biology Markup Language – Flux Balance Constraints*) versión 2. La variante FCB2 tiene la ventaja de que incluye una serie de clases que aportan información sobre parámetros de restricciones de flujo y funciones objetivo, estando orientado a codificar modelos CBM para análisis de flujo (http://sbml.org/Documents/Specifications/SBML_Level_3/Packages/fbc).

Se generaron un total de 34 modelos para la Estrategia 1 (*Tabla 3*) y 2397 modelos para la Estrategia 2 (*Tabla 4*).

Tabla 3. Modelos SBML-FBC2 generados para cada PCG de cada medio (Estrategia 1). Se tienen en cuenta los organismos que crecen juntos en cada experimento.

Medio	Nodo / PCG	Número de organismos diferentes en el experimento	Número de modelos generados por CarveMe
Glucosa	27828 *(Pseudomonas)*	10	5
	35562 (*Enterobacteria*)	7	6
Citrato	28866 (*Pseudomonas*)	4	2
	35562 (*Enterobacteria*)	7	7
Leucina	28853 (*Pseudomonas*)	6	3
	13821 (*Proteobacteria*)	22	11

Tabla 4. Modelos SBML-FBC2 generados para cada PCG de cada medio (Estrategia 2). No se filtró por presencia en los experimentos. Se omitió (*) la generación de modelos para el medio con leucina debido a problemas técnicos (ver discusión).

Medio	Nodo / PCG	Número de organismos en el nodo	Número de modelos generados por CarveMe
Glucosa	27828 *(Pseudomonas)*	1977	1152
	35562 (*Enterobacteria*)	1390	982
Citrato	28866 (*Pseudomonas*)	423	263
	35562 (*Enterobacteria*)	1390	982
Leucina	28853 (*Pseudomonas*)	634	*
	13821 (*Proteobacteria*)	12054	*

2.5. Generación de consensos

Mientras que las dos primeras Estrategias implican analizar múltiples organismos y luego poner los resultados en común, generar consensos es un enfoque opuesto: primero se crean datos comunes y luego se generan resultados. Para hacer un modelo consenso con CarveMe, se planteó en primer lugar la posibilidad de generar modelos individuales para cada organismo seleccionado y posteriormente unirlos en un modelo común. Esta opción se ha llevado a cabo mediante el *script consenso.py*, seleccionando las reacciones comunes a al menos el 80% de los modelos. Sin embargo, consideramos que esta opción no es idónea porque puede dar lugar a problemas de consistencia: cada compuesto de cada reacción del archivo SBML tiene asignado un coeficiente estequiométrico

que está equilibrado con todos los demás. Poner en común las reacciones de varios modelos podría generar contradicciones y desequilibrios, no solo en una misma reacción sino entre varias. Esta conclusión fue secundada por el desarrollador de CarveMe (Machado, comunicación personal).

Así pues, además de generar los modelos consenso a partir de los propios modelos SBML, fueron generados a partir de archivos de anotaciones funcionales creados con eggNOG-mapper. Como se ha introducido anteriormente, eggNOG-mapper crea un archivo por cada genoma de GTDB asignado por Nucmer. Este archivo consiste en una lista de funciones asignadas a cada secuencia codificante de proteínas.

La anotación de cada organismo se realizó con el *script annotate.R*, que llama a su vez a eggNOG-mapper. De nuevo, se trata de un proceso lento que tuvo que ser paralelizado (anotar un genoma toma entre 7 y 8 horas). La creación del consenso en sí la hace otro *script*, en este caso en Python 3, *consenso_EGG.py*, también llamado (opcionalmente) desde *annotate.R*. Este *script* recoge, en primer lugar, todas las reacciones únicas de todos los archivos de anotación dados como datos de entrada y, en segundo lugar, selecciona solamente aquellas que se encuentren en un porcentaje dados de archivos. Los porcentajes seleccionados en este trabajo son del 20%, 50% y 80%. Por último, se guardan estas reacciones en el formato de tabla *.tsv* compatible con CarveMe. Ambos *scripts* se explican en detalle en el Anexo.

En el caso de los consensos de los modelos generados para la Estrategia 1 (es decir, solo aquellos organismos que se observaron en los experimentos de Goldford *et al.* (2018)), se hizo un paso adicional de *gap-filling* con CarveMe para asegurar que los modelos consenso pudieran crecer en su medio correspondiente. Consideramos este paso razonable para estos organismos porque se ha comprobado que son capaces de crecer en ese medio en experimentos reales.

Hacer esto mismo para todos los modelos del nodo (es decir, los generados para la Estrategia 2) sería más *naïve* dado que no se ha comprobado esto para todos ellos. En cualquier caso, en este trabajo nos limitamos a hacer consensos solamente de los modelos creados en la Estrategia 1.

2.6. Análisis con Smetana

Para analizar los modelos creados, se empleó Smetana en primer lugar. Este paso de generación de resultados viene incluido también en *modelado.R*. En segundo lugar, se procesaron los resultados de Smetana con el *script parser.R*, incluido en el Anexo.

2.6.1. Ejecución de Smetana

En cada medio utilizado por Goldford *et al.* (2018) se identificaron dos PCG de profundidad baja a moderada. Como se ha mencionado anteriormente, el objetivo principal del trabajo es entender las posibles interacciones metabólicas entre ambos. Por ello se hicieron dos análisis paralelos para cada pareja inter-nodo (*i. e.* inter-PCG) de cada medio: global y detallado.

En el caso de la Estrategia 1, las únicas parejas estudiadas fueron aquellas que coincidieron en la misma muestra en los experimentos originales. Para cada pareja se ejecutaron varios comandos de Smetana:

- Modo global
- Modo detallado
- Modo detallado, opción *--no-coupling* activada.

La opción --*no-coupling* suprime el cálculo de la SCS (*Tabla 1*). Cuando no se añade esta opción, obtenemos el dato cuantitativo de la dependencia entre los dos organismos analizados además de una lista con todos los intercambios obligatorios para el crecimiento de cada especie. Pero cuando añadimos esta opción, Smetana nos devuelve todos los intercambios posibles, sean necesarios o no. Esto nos permite obtener información sobre la posible cooperación metabólica de cada pareja, aunque sus especies no sean dependientes.

En el caso de la Estrategia 2, se optó por estudiar una muestra de 50000 parejas y ejecutar Smetana únicamente en el modo global y en el detallado con la opción –*no-coupling* para reducir el tiempo de ejecución.

En todos los casos se utilizó la opción --*molweight*. Esta opción afecta directamente a la solución obtenida para cada cálculo y es la recomendada en la documentación de Smetana. Cada par de especies puede interactuar mediante diferentes compuestos y topologías, por lo que hay diferentes soluciones posibles. Esta opción limita las soluciones posibles, priorizando los medios de cultivo no con un menor número de metabolitos sino compuestos de metabolitos más sencillos. Esto da lugar a soluciones más realistas (Machado, comunicación personal).

La opción --*flavor bigg* asegura que se reconozcan correctamente los identificadores, en formato BiGG, de las reacciones de intercambio del modelo.

Por último, en cada ejecución de Smetana hay que especificar el medio de interés, en este caso M9 estándar (con glucosa) o M9 con leucina o citrato en lugar de glucosa (los medios usados por Goldford *et al.* según la publicación original de 2018). Estos medios se definieron y guardaron en el formato descrito en la documentación como M9[glc], M9[leu] y M9[cit] en el archivo *media.tsv*.

2.6.2. Procesado con *parser.R*

Con el *script parser.R* se obtuvo un informe para cada pareja de PCGs en las Estrategias 1, 2 y 3. Los datos incluidos en el informe son:

- Nombre de los archivos vacíos (parejas que contienen modelos que no crecen).
- Nombre de los archivos no vacíos.
- Lista de los 10 metabolitos más intercambiados en ambos sentidos.
- Lista de los 10 metabolitos más cedidos de un PCG a otro.

2.7. Análisis con ReFramed

Se hicieron simulaciones FBA con ReFramed para los modelos creados y las tasas de crecimiento de cada simulación fueron registradas. Los objetivos eran varios:

- Comprobar qué modelos son capaces de crecer sin *gap-filling* en su medio correspondiente y cuánto crecen.
- Comprobar que los modelos consenso crecen en su medio correspondiente y cuánto crecen.
- Comparar las tasas de crecimiento de cada PCG.

Estas simulaciones se automatizaron con el *script RefrFBA.py*. Se analizaron todos los modelos para todas las Estrategias.

3. RESULTADOS

3.1. Estrategia 1: análisis de interacciones metabólicas

La *Tabla 5* muestra una clasificación de las parejas de modelos analizadas con Smetana según la viabilidad de los cálculos de la MIP y la MRO. Los resultados del modo global se incluyen en la *Tabla 6.*

Tabla 5. Viabilidad de los cálculos de la MIP y la MRO por parte de Smetana en la Estrategia 1. Consiste en generar solamente los modelos de aquellos organismos que estaban presentes en la misma muestra de los experimentos originales de *Goldford* et al*. (2018)*. Para cada pareja de modelos, Smetana puede calcular la MRO y la MIP (si ambos modelos son capaces de crecer por su cuenta), solo el MRO (si una de las dos especies no puede crecer por su cuenta) o ninguno (si no crece ninguna, ni siquiera con ayuda del cross-feeding). En el caso del medio M9 con leucina, no crece ningún co-cultivo de modelos.

Medio	**Nodos**	**Parejas con MRO y MIP no disponibles**	**Parejas con MRO calculado pero MIP no disponible**	**Parejas con valores MRO y MIP**
M9[glc]	Nodo 27828 (Pseudomonadaceae) Nodo 35562 (Enterobacteriaceae)	10	0	20
M9[cit]	Nodo 28866 (Pseudomonadaceae) Nodo 35562 (Enterobacteriaceae)	5	3	6
M9[leu]	Nodo 28853 (Pseudomonadaceae) Nodo 13821 (Proteobacteria)	33 (todas)	0	0

Tabla 6. Resumen de los resultados del modo global de Smetana en la Estrategia 1. No se pueden calcular la MIP y la MRO en ninguna de las parejas de leucina.

Medio	MRO			MIP		
	Media	Mínimo	Máximo	Media	Mínimo	Máximo
M9[glc]	0.882	0.815	1	0	0	0
M9[cit]	0.833	0.815	0.889	0	0	0

3.1.1. Glucosa

En el caso del medio M9 con glucosa, todos los informes del modo detallado de Smetana para obtener los metabolitos acoplados al crecimiento están vacíos. Esto es importante porque quiere decir que no se dan intercambios imprescindibles para el crecimiento entre las poblaciones de los PCG que crecen en este medio: en ninguna de las parejas estudiadas se da el caso de que un organismo de un PCG dependa del otro.

Al incluir la opción *--no-coupling*, el modo detallado sí revela, en la mayoría de las parejas estudiadas, una lista de metabolitos que la pareja podría intercambiar para favorecer el crecimiento pero cuyo intercambio no es estrictamente necesario para el mismo (*Tabla 7*, *Figura 6*). En la minoría de casos en los que el informe sigue saliendo vacío, significa que ninguno de los miembros de estas parejas es capaz de crecer en las simulaciones en estos medios.

Tabla 7. Resumen de los resultados del modo detallado (*--no-coupling*) de Smetana en la Estrategia 1. La puntuación *smetana* representa el porcentaje de simulaciones en las que se intercambia ese metabolito. Los identificadores están en formato BiGG (i. e. "gly" → "M_gly_e").

M9[glc]				
10 metabolitos más intercambiados en general		**10 metabolitos más cedidos de un PCG a otro**		
Metabolito	Puntuación *smetana*	Metabolito	Puntuación *smetana*	Nodo donante
acald	0.443	acald	0.486	Nodo35562
ala__L	0.239	acald	0.401	Nodo27828
4abut	0.203	ala__L	0.290	Nodo35562
etha	0.136	4abut	0.235	Nodo27828
ac	0.128	ala__L	0.194	Nodo27828
ptrc	0.115	ac	0.156	Nodo35562
gly	0.114	etha	0.145	Nodo35562
etoh	0.096	4abut	0.140	Nodo35562
glyc3p	0.085	gly	0.139	Nodo35562
acac	0.083	etha	0.134	Nodo27828
M9[cit]				
10 metabolitos más intercambiados en general		**10 metabolitos más cedidos de un PCG a otro**		
Metabolito	Puntuación *smetana*	Metabolito	Puntuación *smetana*	Nodo donante
bz	1	bz	1	Nodo28866
acald	0.255	acald	0.357	Nodo28866
4abut	0.205	4abut	0.320	Nodo28866
etha	0.165	ala__L	0.183	Nodo28866
ala__L	0.160	etha	0.165	Nodo35562
glyc3p	0.138	gly	0.162	Nodo28866
gly	0.134	acald	0.153	Nodo35562
ac	0.117	etoh	0.146	Nodo28866
urea	0.104	ac	0.144	Nodo28866
etoh	0.102	glyc3p	0.139	Nodo35562

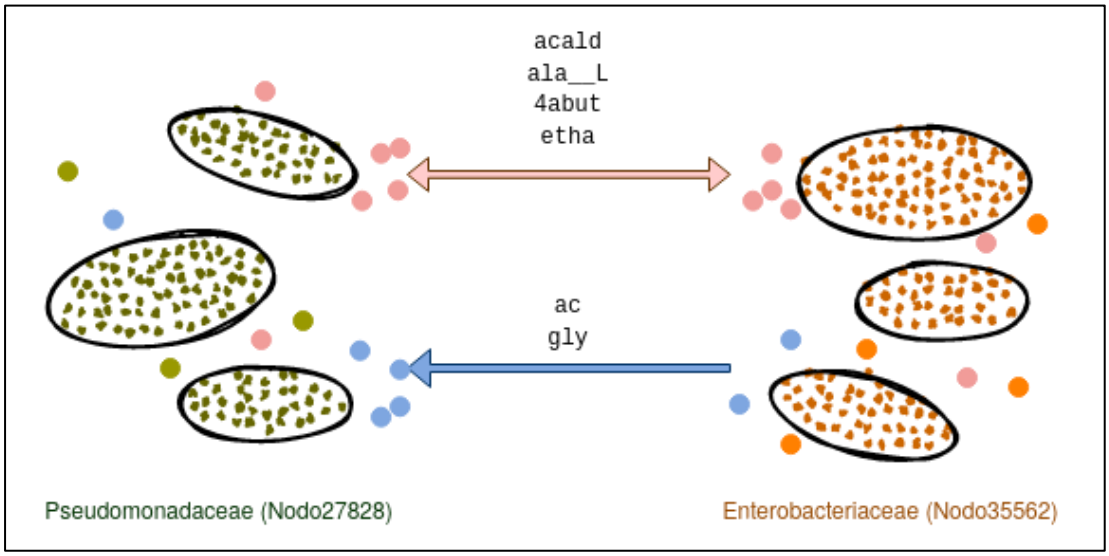

Figura 6. Intercambio de metabolitos entre Pseudomonadaceae y Enterobacteriaceae en M9[glc]. En rosa se indican los metabolitos intercambiados en ambos sentidos; en azul, los metabolitos transferidos de Enterobacteriaceae a Pseudomonadaceae.

En definitiva, según Smetana, en medio *M9[glc]* no hay ninguna pareja interdependiente pero sí hay algunos organismos que no crecen ni por sí mismos (MRO no calculada) ni dependiendo de intercambios con el otro miembro de su pareja (MIP no calculada).

Los valores de MIP obtenidos por el modo global son siempre 0. Por lo tanto, según Smetana, para las parejas que hemos estudiado en este caso no hay ningún metabolito imprescindible intercambiado que puedan tomar de su compañero en lugar de desde un medio completo. El valor medio de la MRO es 0.88. Este valor es bastante elevado (el rango de valores posibles va de 0 a 1), lo cual es coherente con el hecho de que ambos PCGs son cercanos filogenéticamente (clase *Gammaproteobacteria*). La MRO es 1, además, en un caso: la pareja de hojas 333782 y 445257, que corresponden a *Pseudomonas psychrophila* y *Pseudescherichia vulneris*, respectivamente. Un valor de MRO de 1 indica, en teoría, que ambos modelos son capaces de llevar a cabo esencialmente las mismas reacciones químicas. Sin embargo, es posible que los genomas no sean de suficiente calidad o que, ya que esta comparación solo hace referencia a las funciones metabólicas anotadas en el genoma, la anotación no sea buena.

En cuanto a los metabolitos intercambiados entre los dos miembros de la pareja, destaca en primer lugar la posibilidad de un intercambio bidireccional de acetaldehído (*acald*), alanina (*ala_L*) y 4-aminobutanoato (*4abut*), con puntuaciones *smetana* de 0.443, 0.239 y 0.203 respectivamente. Otros compuestos intercambiados en ambos sentidos son el etanol y el acetato, que son de los principales productos de desecho de Enterobacteriaceae (Ferrocino *et al.*, 2017).

3.1.2. Citrato

En el caso del medio M9 con citrato, la mayoría de informes detallados de Smetana sobre los metabolitos acoplados al crecimiento salen vacíos, pero no todos. Es el caso de las parejas 4370747-4454257 y 4419276-4454257 (Greengenes ID), que intercambian los compuestos incluidos en la *Tabla 8*. Se repiten muchos de los compuestos que se intercambiaban en el medio M9[glc], con excepción de la glicina en la segunda pareja. Todos los metabolitos son cedidos al modelo de la familia Enterobacteriaceae desde el modelo de Pseudomonadaceae. Es decir, el crecimiento en M9[cit] de este modelo de una especie de Enterobacteriaceae*, Pseudescherichia vulneris*, es posible gracias a modelos de Pseudomonadaceae.

Los resultados del modo global sugieren que el modelo 3944484 (cuya asignación taxonómica es *Rouxiella sp000257645*) también es dependiente: cuando la MRO es calculada exitosamente, el cálculo de la MIP falla en las dos parejas mencionadas y una más; 4419276 - 3944484. Sin embargo, la lista de metabolitos acoplados al crecimiento para esta pareja sale vacía. No disponemos aún de suficiente información para poder aventurar una explicación para este suceso.

El MRO no puede calcularse para las parejas que incluyen a las hojas 4416113 y 4399988, del nodo 35662 (Enterobacteriaceae). Por tanto, estos modelos no son capaces de crecer en M9[cit] ni solos ni en co-cultivo con los dos modelos de Pseudomonadaceae estudiados. Congruentemente, el modo detallado en modo --*no-coupling* devuelve también archivos vacíos para estas parejas. Los metabolitos intercambiados entre ellas se incluyen en la *Tabla 8*.

Tabla 8. Compuestos intercambiados, según Smetana, en medio M9[cit] entre las parejas de modelos metabólicos 4370747-4454257 y 4419276-4454257. Estos intercambios están acoplados al crecimiento del modelo perteneciente a la familia Enterobacteriaceae.

Pareja 4454257 (nodo 35662, Enterobacteriaceae) – 4370747 (nodo 28866, Pseudomonadaceae)		**Pareja 4454257 (nodo 35662, Enterobacteriaceae) – 4419276 (nodo 28866, Pseudomonadaceae)**	
Compuesto	**Puntuación *smetana***	**Compuesto**	**Puntuación *smetana***
acald	0.65	gly	0.28
ala__D	0.21	acald	0.19
ala__L	0.13	ala__L	0.19
etoh	0.11	etoh	0.15
ac	0.1	ala__D	0.14
for	0.06	ac	0.13
mal__L	0.04	pro__L	0.12
glyc	0.03	glyclt	0.02
leu__L	0.03	ser__D	0.01
succ	0.03		
fum	0.02		
ura	0.02		
val__L	0.02		
asp__L	0.01		
h2s	0.01		
ile__L	0.01		
lac__L	0.01		
rib__D	0.01		

Los valores de MRO son, de nuevo, bastante altos, aunque ligeramente más bajos que los del medio M9[glc] (media de 0.834). La MIP es, de nuevo, siempre 0. Así pues, los intercambios dados entre las dos parejas de la *Tabla 8* dan una ventaja de crecimiento a *Pseudescherichia vulneris*, pero los metabolitos intercambiados no son imprescindibles.

Los metabolitos no necesarios para el crecimiento más intercambiados son los incluidos en la *Tabla 7*. Destacan especialmente los mismos que para el M9[glc]; acetaldehído, 4-aminobutanoato, etanol y alanina. También aparece la glicina, con un valor de puntuación *smetana* parecido al de M9[glc] pero menor que el asignado a la glicina como metabolito acoplado al crecimiento en la pareja 4454257 - 4419276. Sin embargo, el más destacado de todos es el benzoato (*bz*), con una puntuación *smetana* de 1. Este intercambio se da únicamente en la pareja 4419276 - 3944484.

3.1.3. Leucina

Todos los cálculos de MRO y MIP del modo global fallan (*n/a*). Todas las listas de metabolitos del modo detallado salen vacías. Esto es así tanto para los acoplados al crecimiento como para los obtenidos con "*--no-coupling".* Tras repetir una serie de pruebas concluimos que se debe a un fallo técnico que se daba a nivel del *solver:* también fallan el *gap-filling* de CarveMe y las comprobaciones directas de la tasa de crecimiento usando ReFramed, que también dependen de simulaciones FBA.

3.2. Estrategia 1: FBA

Para el medio M9[glc], las tasas medias de crecimiento obtenidas por ReFramed fueron 0.785 ± 0.072 para el nodo 35562 (Enterobacteriaceae) y 0.744 ± 0.072 para el nodo 27828 (Pseudomonadaceae). En el caso del medio M9[cit], la tasa media de crecimiento del nodo 35562 (Enterobacteriaceae) fue 0.553 ± 0.032, y la del nodo 28866 (Pseudomonadaceae) fue 0.546 ± 0.052. Ningún modelo creció en M9[leu]. Como era de esperar, las tasas de crecimiento en general no son muy altas en comparación con las tasas que se dan para un medio completo como el LB (no mostrado). Además, la diferencia entre las tasas de crecimiento de cada PCG es baja.

3.3. Estrategia 2: análisis de interacciones metabólicas

La Estrategia 2 consiste en analizar con Smetana y ReFramed todos los modelos posibles de cada nodo o PCG, se correspondan con especies presentes en los experimentos originales o no. Esto incluye los 2397 modelos (*Tabla 4*) generados a partir de secuencias asignadas por Nucmer con la fiabilidad requerida.

Así pues, en total habría que generar informes de Smetana para 1,131,264 parejas en el caso del M9[glc] y 415,386 en el caso del M9[cit]. Aunque esto es factible (especialmente mediante el uso del método de paralelización integrado en el *script* u otro método alternativo) se trata de un proceso largo: paralelizando en 94 cores, hemos podido generar análisis para 300 parejas cada hora. Por lo tanto, optamos por limitar el análisis con Smetana a una muestra de 50,000 parejas en M9[cit].

Se incluyen a continuación un resumen de los valores de MIP y MRO obtenidos en el modo global de Smetana (*Tablas 9* y *10*).

Tabla 9. Resumen de los valores de MIP y MRO del modo global de Smetana en la Estrategia 2.

Medio	**MRO**			**MIP**		
	Media	**Mínimo**	**Máximo**	**Media**	**Mínimo**	**Máximo**
M9[cit]	0.867	0.690	1	0.006	0	2

Tabla 10. Viabilidad de los cálculos de la MIP y la MRO por parte de Smetana en la Estrategia 2. Esta estrategia consiste en analizar con Smetana y ReFramed todos los modelos posibles de cada nodo o PCG, se correspondan con especies presentes en los experimentos originales o no. Se realizó este análisis para el medio con citrato (M9[cit]).

Medio	**Nodos**	**Parejas con MRO y MIP no disponibles**	**Parejas con MRO calculado pero MIP no disponible**	**Parejas con valores MRO y MIP**
M9[cit]	Nodo 28866 (Pseudomonadaceae) Nodo 35562 (Enterobacteriaceae)	12906 (25.81%)	6074 (12.15%)	31020 (62.04%)

Los valores de MIP disponibles no son siempre 0 como en la Estrategia 1, sino que es 1 en 166 de las parejas e incluso 2 en 9 de ellas. Esto implica que en estas parejas uno de los modelos aprovecha uno o dos metabolitos producidos por el otro, reduciendo su dependencia del medio.

Los valores medios de MRO son muy cercanos a los de la Estrategia 1, aunque con un mínimo más bajo y un máximo de 1. El MRO de 1 aparece en 2200 parejas, que incluyen 137 modelos de Enterobacteriaceae y 192 de Pseudomonadaceae. El MIP suele ser 0 (30845 parejas).

Como se puede observar en la *Tabla 10*, hay 6,074 parejas para las que el MRO está disponible pero el MIP no. Por lo tanto, en todas estas parejas hay algún modelo que no es capaz de crecer por sí mismo pero sí crece gracias a los intercambios con el otro miembro de la pareja.

Contando los modelos de cada nodo que aparecen en estas parejas, podemos ver que, por ejemplo, el modelo de Pseudomonadaceae correspondiente a la hoja 593619 da este resultado con 204 de los 230 modelos de Enterobacteriaceae con los que ha sido emparejado en este análisis, por lo que podemos deducir que es este modelo el que tiene problemas para crecer. Del mismo modo, entre los modelos de Enterobacteriaceae hay varios que dan lugar a un fallo del cálculo de la MIP con la inmensa mayoría de los 198 modelos de Pseudomonadaceae analizados (por ejemplo, 195 para las hojas 103166 y 236285), que también podemos suponer que no son capaces de crecer por sí solos pero sí en co-cultivo.

Llama la atención el caso de las hojas 103166 y 236285 ya mencionadas: tienen una MRO distinta a 0 en 195 casos, por lo que solo hay tres modelos de Pseudomonadaceae con los que no son capaces de crecer en co-cultivo. Sin embargo, hemos comprobado que hay 24 modelos de Pseudomonadaceae que tienen generalmente problemas para crecer (>100 parejas con MIP no disponible, apareciendo 10 de ellos en >150 de estas parejas).

Así pues, parece que sí se están dando ciertos casos en los que ninguno de los dos miembros de la pareja puede crecer solo, pero juntos sí pueden: interdependencia. Efectivamente, ejecutando Smetana para 236285 (Enterobacteriaceae) y 593619 (Pseudomonadaceae) sin la opción *--no-coupling* vemos que hay intercambio de metabolitos acoplados al crecimiento en ambos sentidos. Los más importantes (puntuación *smetana* mayor) son la alanina (0.52) y el acetaldehído (0.29) desde Enterobacteriaceae hasta Pseudomonadaceae y el ácido N-acetil-D-glucosamina(anhidro)N-acetilmurámico (ANHGM) (1) en sentido contrario.

A continuación, se incluyen en la *Tabla 11* los metabolitos más intercambiados de acuerdo con los informes del modo detallado. En primer lugar, podemos ver que el benzoato, el acetaldehído y la alanina están entre los diez metabolitos más intercambiados al igual que se vio en la Estrategia 1. Sin embargo, esta vez vemos el intercambio de benzoato ocurre en ambos sentidos por igual, no solo de Pseudomonadaceae hacia Enterobacteriaceae. El intercambio de acetaldehído también es mutuo.

En segundo lugar, *parser.R* revela que el transporte de ANHGM va principalmente desde Enterobacteriaceae hacia Pseudomonadaceae. Este transporte se da, siempre en el mismo sentido, en 211 parejas de un total de 36,982 parejas con informes no vacíos.

En tercer lugar, aparecen más metabolitos que no se habían visto en los modelos de los organismos del experimento original. Es el caso del inositol, la timina y el formaldehído, cedidos principalmente por Enterobacteriaceae; el pantotenato, cedido por Pseudomonadaceae; el tolueno y la fosfoserina.

Tabla 11. Resumen de los resultados del modo detallado (*--no-coupling*) de Smetana en la Estrategia 2 para el medio M9[cit]. La puntuación *smetana* representa el porcentaje de cultivos simulados en los que se da el intercambio de ese metabolito. El identificador de cada metabolito se corresponde con el formato BiGG (i. e. "gly" → "M_gly_e").

10 metabolitos más intercambiados en general		**10 metabolitos más cedidos de un PCG a otro**		
Metabolito	Puntuación *smetana*	Metabolito	Puntuación *smetana*	Nodo donante
anhgm	1	anhgm	1	Nodo35562
inost	0.936	inost	0.940	Nodo35562
thym	0.781	thym	0.786	Nodo35562
pnto__R	0.690	pnto__R	0.689	Nodo28866
bz	0.381	fald	0.432	Nodo35562
fald	0.348	acald	0.396	Nodo28866
acald	0.345	bz	0.395	Nodo35562
tol	0.283	bz	0.365	Nodo28866
pser	0.248	his__L	0.305	Nodo28866
his	0.196	acald	0.294	Nodo35562

3.4. Estrategia 2: FBA

En esta ocasión, las tasas de crecimiento medias obtenidas por ReFramed para el medio M9[glc] fueron 0.732 ± 0.069 para el nodo de Pseudomonadaceae (27828) y 0.820 ± 0.031 para el nodo de Enterobacteriaceae (35562). En el caso del medio M9[cit], la tasa media de crecimiento del nodo de Pseudomonadaceae (28866) fue 0.549 ± 0.032, y la del nodo de Enterobacteriaceae (35562) fue 0.567 ± 0.025.

Las tasas de crecimiento medias obtenidas son más fiables que las de la Estrategia 1 porque la muestra utilizada es mayor, pero vemos que son muy similares, así que podemos considerar que eran representativos de los PCGs. La desviación típica también es parecida, aunque ligeramente menor.

De 1,152 modelos creados para el nodo 27828, 427 no crecieron en el FBA. De los modelos del nodo 28866, hubo 67 de 263 que no crecieron. Para el nodo 35562, el crecimiento fue 0.0 para 131 de 982 modelos en M9[glc] y para 142 de 982 en M9[cit]. Podemos ver que las proporciones en la Estrategia 1 eran similares y por tanto eran, también, representativas.

3.5. Estrategia 3: análisis de interacciones metabólicas

Como parte de los objetivos del proyecto, se han derivado modelos metabólicos consenso de cada PCG. Hemos explorado dos posibilidades a la hora de generarlos: a partir de los modelos individuales intra-PCG directamente y a partir de las anotaciones intra-PCG (generando un modelo a partir de la anotación consenso). A continuación se incluyen los resultados de los análisis con Smetana de estos modelos consenso y las impresiones que se derivan de los mismos.

Al generar modelos consenso a partir de las anotaciones funcionales obtenidas con eggNOG-mapper de los modelos de la Estrategia 1, se han seleccionado distintos grados de flexibilidad. Esta flexibilidad viene determinada por el porcentaje de ficheros de entrada en los que se indique que debe estar contenida una anotación funcional para que esta sea incluida en el modelo consenso (Anexo). Asimismo, los consensos se han hecho de dos formas distintas en todos los casos: incluyendo todos los modelos o incluyendo solo aquellos que crecían según el FBA.

No se han conseguido generar consensos para porcentajes del 50% o superiores cuando se han excluido aquellos modelos de la Estrategia 1 que no crecían según el análisis previo FBA. La razón

de esto es que para esos casos no había ninguna reacción que se encontrara a la vez en la mitad de los modelos o más.

Los resultados de los análisis globales con Smetana se resumen en la *Tabla 12*.

Tabla 12. Resumen de los resultados (MRO y MIP) de los análisis del modo global de Smetana para cada modelo consenso generado.

Medio			**M9[glc]**		**M9[cit]**	
Métrica			**MRO**	**MIP**	**MRO**	**MIP**
Consenso SBML	Todos los modelos		0.857	0	0.889	0
	Excluyendo aquellos que no crecen según Smetana		1	0	0.857	0
Consensos eggNOG	Todos los modelos	20%	0.889	0	0.786	1
		50%	0.966	0	0.897	1
		80%	0.966	0	0.897	0
	Excluyendo aquellos que no crecen según FBA	20%	0.8890	0	1	0

Todos los informes del modo detallado de Smetana indican que no hay ningún intercambio, en ningún caso, necesario para el crecimiento. Los casos aislados de la Estrategia 1 donde sí que existía dicho intercambio necesario no quedan, pues, representados en el consenso.

En los informes del modo global, los valores de la MRO son cercanos a los observados en las Estrategias 1 y 2 (*Tablas 6* y *9*), aunque se desvían ligeramente (± 0.1) de los rangos observados. La MIP es siempre 0, como en los análisis de la Estrategia 1 cuyos modelos están incluidos en estos consensos, excepto en los informes de las parejas de modelos al 20% y 50% del medio M9[cit], donde es 1. Esto implica que estos consensos, que incluyen más reacciones que el del filtro del 80%, contienen alguna reacción o reacciones que permiten a uno de los PCGs miembros de la pareja aprovechar un metabolito producido por el otro, reduciendo su dependencia del medio.

Los metabolitos más intercambiados en el medio M9[cit] varían ligeramente según el porcentaje, pero no mucho: en los consensos al 50% y 80%, los más transportados de Pseudomonadaceae a Enterobacteriaceae son el alfa-cetoglutarato (puntuación *smetana* de 0.42 y de 0.5 respectivamente) y el succinato (0.27, 0.39); asimismo, los más transportados en sentido contrario son el acetaldehído (0.29, 0.49) y el sulfato de hidrógeno (0.28, 0.15). En el consenso al 20%, estos metabolitos también son de los más intercambiados pero no están en el mismo orden, mezclándose con el glutamato (0.35) y la glicerol-3-fosfoetanolamina (0.11). Tampoco hay mucha diferencia entre este último informe y el correspondiente a los consensos de solo aquellos modelos que crecían según el FBA. La más notable es que el intercambio de acetaldehído no es tan relevante en este último (0.04, frente a 0.4).

Los metabolitos más intercambiados en M9[glc] (para los consensos que no excluían ningún modelo) varían bastante entre porcentajes y, salvo la glicerol-3-fosfoetanolamina en los resultados al 20% y 50% (puntuaciones de 0.62 y 0.72 respectivamente) y el acetaldehído, tienen puntuaciones relativamente bajas. Este último se mueve en ambos sentidos por igual en los consensos al 20% y predominantemente hacia el PCG de Enterobacteriaceae en los del 50%, pero no aparece en el análisis correspondiente a los modelos consenso al 80%.

Cuanto mayor es el porcentaje exigido a la hora de realizar el consenso, menos intercambios hay. Esto se cumple también en el caso del M9[cit]. Así pues, se observa que el efecto de filtrar reacciones desemboca en una disminución de los intercambios.

Comparando con los metabolitos más intercambiados según se vio en la Estrategia 1, vemos que uno de los más intercambiados, el acetaldehído, también aparece. Sin embargo, el intercambio de alanina no destaca o incluso no aparece en los consensos de la Estrategia 3, aunque era uno de los tres compuestos más intercambiados en ambos medios. Asimismo, la glicerol-3-fosfoetanolamina, por ejemplo, no aparecía en la Estrategia 1.

También se obtuvieron consensos a partir de los propios modelos SBML con *consenso.py*, aunque, como ya se ha mencionado, consideramos este método menos fiable *a priori*.

Como se ve en la *Tabla 12*, los MRO están en un rango similar excepto el de M9[glc], que llega a 1. No se reporta tampoco ningún metabolito necesario para el crecimiento.

Respecto a los metabolitos intercambiados, primero comentaremos los consensos obtenidos filtrando según crecimiento por FBA. De los metabolitos no necesarios para el crecimiento pero potencialmente intercambiados, destaca de nuevo el acetaldehído en primer lugar, dato en que coinciden todos los métodos. Estos consensos, curiosamente, sí destacan el intercambio de alanina (reportado en las dos Estrategias anteriores), además del de acetato (que se reportaba en los análisis de los modelos presentes en el experimento). También aparecen la leucina y el sulfuro de hidrógeno de los consensos de eggNOG-mapper.

Los metabolitos intercambiados entre los consensos generados sin filtrar por crecimiento son muy parecidos: destacan con diferencia el acetaldehído, la alanina y el acetato.

3.6. Estrategia 3: FBA

La *Tabla 13* incluida a continuación recoge las tasas de crecimiento medias de los modelos consenso generados. Se incluyen también las tasas medias de crecimiento obtenidas en las Estrategias 1 y 2.

Tabla 13. Comparación de las tasas de crecimiento obtenidas durante este trabajo para cada PCG. Se incluyen las tasas medias de los modelos estudiados individualmente en las Estrategias 1 y 2, además de las tasas correspondientes a todos los consensos generados. Para los consensos obtenidos a partir de las anotaciones de eggNOG-mapper (*consenso_EGG.py*) se han utilizado tres valores distintos a la hora de seleccionar reacciones. También hay una tercera serie de tasas de crecimiento, que se corresponde a los modelos obtenidos a partir del consenso directo de modelos SBML (*consenso.py*). Los consensos se han hecho de dos formas distintas en todos los casos: incluyendo todos los modelos o incluyendo solo aquellos que crecían en FBA (*: todos los modelos crecían según FBA).

Medio			**M9[glc]**		**M9[cit]**	
PCG (nodo)			**35562**	**27828**	**35562**	**28866**
Consenso SBML	Todos los modelos (80%)		0.716	0.560	0.439	0.507
	Excluyendo aquellos que no crecen según Smetana (80%)		0.671	0.560	0.511	0.499
	Todos los modelos	20%	0.694	0.507	0.453	0.106

Consensos eggNOG		50%	0.320	0.514	0.435	0.114
		80%	0.398	0.456	0.282	0.114
	Excluyendo aquellos que no crecen según FBA	20%	0.332	0.586*	0.305	0.064*
		50%	-	-	-	0.067
		80%	-	-	-	0.057
Media Estrategia 1			0.785	0.744	0.553	0.546
Media Estrategia 2			0.820	0.732	0.567	0.549

En el caso de los consensos generados a partir de las anotaciones de eggNOG-mapper, como se ha descrito en la sección anterior, se han probado diferentes grados de flexibilidad: la inclusión en el consenso de aquellas reacciones presentes en el 20%, 50% o 80% de los modelos de entrada. Se puede apreciar que las tasas de crecimiento no son deterministas: por ejemplo, los modelos marcados con un asterisco en la tabla son exactamente iguales a los otros modelos del 20% (todos los modelos de este nodo mostraban crecimiento, así que ninguno se dejó fuera), pero vemos que las tasas obtenidas son ligeramente distintas.

Los consensos eggNOG que tienen una tasa de crecimiento más similar a la media de los modelos generados en las Estrategias 1 y 2 son aquellos con un porcentaje menos estricto. Destaca el caso del nodo 28866 (Pseudomonadaceae) del medio M9[cit], que es mucho menor en todos los consensos que las medias de los modelos estudiados individualmente en las Estrategias 1 y 2.

También se incluyen en la *Tabla 12* las tasas de crecimiento de los consensos generados con *consenso.py*, es decir, directamente desde los modelos SBML. A pesar de que, como ya se adelantó en la sección de Metodología, no consideramos fiable *a priori* este método de generar consensos, los valores de crecimiento de estos modelos son los más similares a las tasas medias de las Estrategias 1 y 2, incluso aplicando un filtro del 80%.

Ningún modelo consenso de los incluidos en la *Tabla 12* fue capaz de crecer sin hacerle *gap-filling* previamente.

4. DISCUSIÓN

Los objetivos de este trabajo fueron dos: contrastar la hipótesis de cohesión ecológica intra-grupo y modularidad de los PCGs en la construcción de una comunidad mediante el análisis de un caso real (Goldford *et al.*, 2018) y desarrollar *scripts* para automatizar dicho análisis. A continuación, discutiremos la consecución de estos objetivos en base a los resultados obtenidos.

4.1. Conclusiones biológicas

4.1.1. Modularidad y cohesión ecológica intra-PCG

Se han detectado dos PCGs en cada medio. Estos PCGs coinciden, para los medios M9[glc] y M9[cit], con los grupos taxonómicos observados en el trabajo original de Goldford *et al.* (2018): Enterobacteriaceae y Pseudomonadaceae.

En el caso del M9 con leucina veíamos un PCG de Pseudomonadaceae y otro más amplio afiliado al filo Proteobacteria. Sin embargo, los resultados derivados de dicho medio no se han podido estudiar a fondo por problemas técnicos ya avanzados y que se discutirán más adelante.

El presente análisis se centra en las funciones metabólicas y los consecuentes intercambios de compuestos entre cada pareja de PCGs. Un PCG es una porción de la filogenia presente en cada instancia diferente de un mismo ecosistema, en este caso cada réplica del experimento de Goldford *et al.* (2018), para la cual se predice un conjunto de características filogenéticas conservadas que lo relacionan con un nicho funcional concreto (Aguirre de Cárcer, 2019). Se ha podido observar una clara cohesión funcional dentro de cada PCG en varios puntos. En primer lugar, las tasas de crecimiento de cada modelo son similares a lo largo de todo el nodo/PCG, con una desviación estándar de un 10% como máximo en la Estrategia 1, que era incluso más baja en la Estrategia 2. En segundo lugar, y ya teniendo en cuenta las interacciones inter-PCG, en los resultados de Smetana para todas las parejas estudiadas en la Estrategia 1 y la 2, se repiten varios compuestos entre los metabolitos más intercambiados y se obtienen unos valores de MRO y MIP muy cercanos entre sí.

Asimismo, las tasas de crecimiento medias de los dos PCG de cada medio son similares entre sí, lo que explica la estabilidad de su coocurrencia en unos experimentos de dilución seriada como los estudiados.

4.1.2. Interacciones metabólicas entre ambos PCGs

En los ecosistemas estudiados (medios) hay dos PCGs distintos. Cuando dos especies o grupos de especies son antagónicas, no pueden convivir a menos que haya diferentes parches en el medio, en otras palabras, que cada uno se instale en una sección diferente en el espacio (Cordero and Datta, 2016). En este caso se trata de un cultivo en un medio artificial homogéneo y líquido, así que en principio no debería haber diferencias espaciales. Es decir, podemos concluir que estos PCGs no están en una situación de antagonismo.

Sin embargo, hemos visto unos valores de superposición de recursos metabólicos, dada por la MRO media (cercana a 1), muy altos para múltiples parejas de modelos inter-PCG, incluso de modelos que representan organismos en el experimento real (Estrategia 1). En principio, esto implica una alta probabilidad de competir por los mismos metabolitos: ambos PCGs necesitan consumir metabolitos similares para crecer. A pesar de ello, estamos ante un ejemplo de coexistencia de dos PCGs.

Goldford *et al.* (2018) propusieron una explicación para la presencia de estas dos familias bacterianas que encaja con esta idea. Como se ha visto en la mayoría de nuestros resultados además de en los experimentos originales, especies/modelos pertenecientes a ambas PCG son capaces de crecer por sí mismas en dichos medios. Los autores defienden que cada familia conserva una función metabólica distinta, pero a la vez similar a la de la otra familia, que le permite superar los filtros abióticos necesarios para permanecer en la comunidad. Enterobacteriaceae conserva a nivel de familia un eficaz sistema de fosfotransferasas y Pseudomonadaceae posee a su vez transportadores unidos a ATP. Se trata de dos sistemas diferentes pero muy eficaces a la hora de tomar glucosa del medio (Klingenberg, 1987). Una explicación equivalente podría darse para el citrato: ambos PCGs están utilizando el mismo metabolito principal para crecer, glucosa o citrato.

Así pues, según Goldford *et al.* (2018) estos dos grupos o PCGs están siendo seleccionados por la misma característica, por lo que están ocupando el mismo nicho. La razón por la que no llega a haber competición entre ambos es que son funcionalmente muy similares, idea secundada por nuestras propias observaciones, que incluyen las tasas de crecimiento y el intercambio de

metabolitos que se da indistintamente en un sentido que en otro. Además, los parecidos ratios de crecimiento que observamos son congruentes con que estos dos grupos aguanten múltiples generaciones de cultivo y dilución sin que ninguno de los dos se pierda. El hecho de que parezca haber cierta dependencia inter-PCG mutua, no unilateral, para algunos modelos estudiados no presentes en el experimento original (Estrategia 2) también apunta en esta dirección, aunque es importante tener en cuenta que estos modelos pueden estar incompletos y no reflejar completamente la realidad experimental.

Por otro lado, Estrela *et al.* (2020), que han llevado a cabo simulaciones CAFBA con modelos curados de Enterobacteriaceae y Pseudomonadaceae, defienden que la coexistencia de estas dos familias se debe a que Pseudomonadaceae tiene una ventaja selectiva, no por su consumo de glucosa sino por ser capaz de crecer eficientemente en los productos de desecho de Enterobacteriaceae, principalmente succinato, acetato y lactato. Se basaron en experimentos propios, que demostraron que Pseudomonadaceae crece más rápido en succinato y el doble de rápido en acetato.

En los resultados de nuestros análisis con Smetana hemos detectado, efectivamente, intercambios de acetato (en los modelos correspondientes al experimento original y también en el análisis de los consensos) y succinato con una alta puntuación *smetana* entre estos últimos. No obstante, se vio que el acetato era transportado predominantemente desde los nodos de Pseudomonadaceae hacia los de Enterobacteriaceae, no al revés. El metabolito más intercambiado en nuestros resultados era siempre el acetaldehído, en ambos medios y en ambos sentidos según los resultados de las Estrategias 1 y 2, y cedida por Enterobacteria a Pseudomonadaceae según los de la 3.

Es concebible, para aclarar esta cuestión, realizar un FBA adicional con un nuevo medio que incluya los subproductos metabólicos de Enterobacteriaceae y otro que incluya los de Pseudomonadaceae. El objetivo sería ver si los modelos son viables en este nuevo medio y si sus tasas de crecimiento están más cercanas a las observadas experimentalmente.

4.2. Consideraciones técnicas

Los resultados de este trabajo han evidenciado una serie de limitaciones técnicas. Las comentaremos a continuación.

En primer lugar, ningún modelo creado ha sido capaz de crecer en M9[leu]. Posteriormente, fue confirmado por el desarrollador de CarveMe que los modelos universales que utiliza esta herramienta no son capaces de utilizar la leucina, por lo que tendrá que actualizarlos. Por lo tanto, tampoco es posible hacer ningún tipo de *gap-filling*, proceso que añade reacciones pertenecientes a este modelo universal, ni ver ningún tipo de crecimiento en FBA. Efectivamente, todos los análisis con leucina que se intentó llevar a cabo daban resultados congruentes con esta idea. Esto ha limitado el análisis objetivo de este trabajo, ya que solo se han podido estudiar dos de los tres medios deseados.

Asimismo, hemos comprobado que algunos modelos no son capaces de crecer en ninguno de los medios específicos derivados de M9. La contradicción está en que esto ocurre para modelos con organismos filogenéticamente cercanos que se ha comprobado experimentalmente que son capaces de crecer en estos medios. Esto, más que darnos información sobre los organismos reales, nos está informando sobre la calidad de los datos genómicos utilizados. CarveMe podría estar fallando en varios puntos clave: en primer lugar, CarveMe podría estar partiendo de anotaciones funcionales erróneas, por ejemplo asignando a un gen la función de un gen parálogo, un problema bastante común (Huerta-Cepas *et al.*, 2017); en segundo lugar, podría haber genes que directamente no tengan una función concreta asignada en las bases de datos; en tercer lugar, puede

haber incorrecciones debidas a que el modelo universal es también un modelo incompleto (esto se evidencia en que no es capaz de utilizar leucina), y en cuarto lugar, y más probable, puede que a pesar de haber empleado una base de datos de genomas supuestamente curados y de alta calidad (GTDB, https://data.ace.uq.edu.au/public/gtdb/data/releases/release95/95.0/) haya casos en los que la calidad no sea la suficiente; es más que posible que en muchos casos falten partes del genoma y esto lleve a la falta de reacciones en los modelos.

En cuanto a los consensos, si bien los resultados demostraron la viabilidad de la opción de producir modelos consenso a partir de modelos individuales, este método nos genera dudas, compartidas además con el desarrollador de Smetana y ReFramed, sobre que se puedan dar efectos adversos (impredecibles *a priori*) relacionados con la consistencia interna del modelo generado.

La segunda vía, anotación de los genomas intra-nodo y posterior generación de una anotación consenso de la cual derivar el modelo metabólico, se presentaba como la opción más fiable. Sin embargo, aunque eggNOG-mapper utiliza una base de datos que utiliza grupos de ortólogos para evitar el problema de los parálogos derivado de las búsquedas por homología, existe la posibilidad de que pueda asignar un número menor de funciones metabólicas. Esto se ve reflejado en las tasas de crecimiento de los consensos generados a partir de sus anotaciones, que son ligeramente más bajas que las medias de las Estrategias 1 y 2 y mucho más bajas en el caso del nodo 28866 de M9[cit].

Por último, aunque sí crecían en simulaciones con medio completo, ningún modelo consenso ha sido capaz de crecer en los medios estudiados sin hacer *gap-filling* previo, ni siquiera cuando se seleccionaban como hojas de entrada solo aquellas que ya crecían. En el caso de los consensos creados por la primera vía, a partir de modelos SBML, esta incongruencia es una prueba más de los efectos secundarios no deseados que puede tener este enfoque. En el caso de la segunda vía, los archivos de entrada eran archivos de anotaciones de los genomas. Por tanto, una posible explicación es que las reacciones que permiten a los modelos individuales crecer no están siendo incluidas como anotaciones por parte de eggNOG-mapper, bien porque 1) no están en la base de datos de eggNOG o bien porque 2) no están en el genoma original. En este último caso, puede ser que las reacciones que permitían el crecimiento sí aparecieran en los modelos individuales finales porque CarveMe las incluía a partir de su modelo universal, cosa que no hace con el modelo consenso por diferencias en el *input* que no sabemos explicar *a priori*. Sea cual sea la causa, sería necesario mejorar nuestra comprensión del sistema de anotaciones eggNOG para poder rediseñar una estrategia de creación de modelos consenso adecuada.

A la vista de las incongruencias ya comentadas, nuestros modelos *naïve* no son tan fiables como podrían ser los obtenidos a partir de un modelo curado. De hecho, Estrela *et al.* (2020) usaron el mismo *solver* que se ha usado en este trabajo, CPLEX, pero con nuestros modelos se obtienen ratios de tasas de crecimiento diferentes, aunque se use CAFBA con los mismos parámetros. No obstante, la sencillez de la creación y análisis de modelos *naïve* hace posible aplicar este enfoque a otros campos de estudio sin la necesidad de crear modelos curados ni la compleja recopilación de información que ello conlleva.

5. CONCLUSIONES Y PERSPECTIVAS FUTURAS

El enfoque de este trabajo es *naïve*: se han utilizado modelos no curados, generados automáticamente a partir de datos genómicos. La única modificación realizada ha sido el *gap-filling* a los consensos, también automático. A pesar de su sencillez, este enfoque nos ha permitido obtener información biológicamente relevante.

De cara al futuro, nos queda optimizar el *framework* propuesto. Por lo pronto, las Estrategias 1 y 2 presentan la ventaja de ser más rápidas que crear consensos con eggNOG-mapper: la generación de un modelo con CarveMe toma dos minutos mientras anotar un genoma toma más de seis horas. Por otro lado, la Estrategia 1 tiene la ventaja de que, al seleccionar los modelos de especies que se ha comprobado experimentalmente que son capaces de crecer, permite hacer un rápido paso extra de curado, el *gap-filling*. Además, al crear un mucho menos número de modelos, es más rápida. Sin embargo, la Estrategia 2 (incluir todas las especies del PCG posibles) aporta información extra en el análisis del modo detallado de Smetana, ya que al trabajar con una muestra más amplia nos permite observar más casos de intercambios acoplados al crecimiento, más metabolitos relevantes e incluso indicios de interdependencia. Que la muestra sea más amplia también da una mayor fiabilidad a los valores de crecimiento obtenidos por FBA. La generación de consensos, aunque parece interesante, presenta los problemas ya comentados tanto con el enfoque directo como mediante anotaciones de eggNOG-mapper. En el caso del enfoque directo, aunque sea mucho más rápido por no realizar anotación, puede generar problemas inesperados.

Durante las tres Estrategias, hemos observado que los PCGs co-ocurrentes en ambos medios son suficientemente similares entre sí en su tasa de crecimiento como para que uno de ellos no supere (y elimine) al otro a lo largo de los 12 pases, lo que es coherente con la hipótesis original de Goldford *et al.* (2018). Aunque la información aportada por los metabolitos intercambiados entre PCGs es limitada por la calidad de las anotaciones funcionales, puede ser biológicamente relevante. Además, los resultados indican que los consensos a nivel de PCG son en principio interesantes para realizar estudios del metabolismo.

En conclusión, en este trabajo se ha propuesto un enfoque de modelado metabólico *naïve* de comunidades microbianas simples basado en observaciones experimentales de señal filogenética y se ha demostrado que es capaz de alcanzar conclusiones similares a las obtenidas experimentalmente, además de proporcionar información adicional que podría ayudar a explicar las causas últimas de los patrones de diversidad observados. El *framework* computacional establecido y el conjunto de *scripts* diseñados pueden utilizarse con cualquier otro ecosistema microbiano. Si bien en la actualidad está limitado al análisis de modelos por parejas, es perfectamente posible escalar su arquitectura para el estudio de múltiples modelos, incluso pertenecientes a más de dos nodos diferentes.

BIBLIOGRAFÍA

Aguirre de Cárcer, Daniel. 2019. "A Conceptual Framework for the Phylogenetically Constrained Assembly of Microbial Communities." *Microbiome* 7(1): 142.

Anderson, J L, R J Edney, and K Whelan. 2012. "Systematic Review: Faecal Microbiota Transplantation in the Management of Inflammatory Bowel Disease." *Alimentary Pharmacology & Therapeutics* 36(6): 503–16.

Bauer, Eugen et al. 2017. "BacArena: Individual-Based Metabolic Modeling of Heterogeneous Microbes in Complex Communities" ed. Costas D Maranas. *PLoS Computational Biology* 13(5): e1005544.

Brunner, J D, and N Chia. 2019. "Metabolite-Mediated Modelling of Microbial Community Dynamics Captures Emergent Behaviour More Effectively than Species-species Modelling." *Journal of The Royal Society Interface* 16(159): 20190423.

de Cárcer, Daniel Aguirre. 2020. "Experimental and Computational Approaches to Unravel Microbial Community Assembly." *Computational and Structural Biotechnology Journal* 18: 4071–81.

Cordero, Otto X, and Manoshi S Datta. 2016. "Microbial Interactions and Community Assembly at

Microscales." *Current Opinion in Microbiology* 31: 227–34.

Craven, Laura J et al. 2017. "Extended Screening Costs Associated With Selecting Donors for Fecal Microbiota Transplantation for Treatment of Metabolic Syndrome-Associated Diseases." *Open Forum Infectious Diseases* 4(4).

Dessaux, Yves, Catherine Grandclément, and Denis Faure. 2016. "Engineering the Rhizosphere." *Trends in Plant Science* 21(3): 266–78.

Dias, Oscar, Miguel Rocha, Eugénio C Ferreira, and Isabel Rocha. 2015. "Reconstructing Genome-Scale Metabolic Models with Merlin." *Nucleic Acids Research* 43(8): 3899–3910.

Enke, Tim N et al. 2019. "Modular Assembly of Polysaccharide-Degrading Marine Microbial Communities." *Current Biology* 29(9): 1528--1535.e6.

Estrela, Sylvie, Alicia Sanchez-Gorostiaga, Jean C C Vila, and Alvaro Sanchez. 2020. "Nutrient Dominance Governs the Assembly of Microbial Communities in Mixed Nutrient Environments."

Ferrocino, Ilario et al. 2017. "Shotgun Metagenomics and Volatilome Profile of the Microbiota of Fermented Sausages" ed. Christopher A Elkins. *Applied and Environmental Microbiology* 84(3).

Goldford, Joshua E et al. 2018. "Emergent Simplicity in Microbial Community Assembly." *Science* 361(6401): 469–74.

Gonze, Didier, Katharine Z Coyte, Leo Lahti, and Karoline Faust. 2018. "Microbial Communities as Dynamical Systems." *Current Opinion in Microbiology* 44: 41–49.

Henry, Christopher S et al. 2010. "High-Throughput Generation, Optimization and Analysis of Genome-Scale Metabolic Models." *Nature Biotechnology* 28(9): 977–82.

Ho, Adrian, D. Paolo Di Lonardo, and Paul L.E. Bodelier. 2017. "Revisiting Life Strategy Concepts in Environmental Microbial Ecology." *FEMS microbiology ecology*.

Hoek, Tim A et al. 2016. "Resource Availability Modulates the Cooperative and Competitive Nature of a Microbial Cross-Feeding Mutualism" ed. Nathalie Balaban. *PLoS Biology* 14(8): e1002540.

Huerta-Cepas, Jaime et al. 2017. "Fast Genome-Wide Functional Annotation through Orthology Assignment by eggNOG-Mapper." *Molecular Biology and Evolution* 34(8): 2115–22.

Karp, Peter D et al. 2015. "Pathway Tools Version 19.0 Update: Software for Pathway/Genome Informatics and Systems Biology." *Briefings in Bioinformatics* 17(5): 877–90.

Klingenberg, P. 1987. "G. Gottschalk: Bacterial Metabolism. 2. Aufl. 359 Seiten, 204 Abb. Springer-Verlag, New York, Berlin, Heidelberg, Tokyo 1986. Preis: 98,— DM." *Food / Nahrung*.

Kurtz, Stefan et al. 2004. "No Title." *Genome Biology* 5(2): R12.

Li, Simone S et al. 2016. "Durable Coexistence of Donor and Recipient Strains after Fecal Microbiota Transplantation." *Science* 352(6285): 586–89.

Lieven, Christian et al. 2020. "MEMOTE for Standardized Genome-Scale Metabolic Model Testing." *Nature Biotechnology*.

Machado, Daniel, Sergej Andrejev, Melanie Tramontano, and Kiran Raosaheb Patil. 2018. "Fast Automated Reconstruction of Genome-Scale Metabolic Models for Microbial Species and Communities." *Nucleic Acids Research* 46(15): 7542–53.

Mendes, R et al. 2011. "Deciphering the Rhizosphere Microbiome for Disease-Suppressive Bacteria." *Science* 332(6033): 1097–1100.

Momeni, Babak, Li Xie, and Wenying Shou. 2017. "Lotka-Volterra Pairwise Modeling Fails to Capture Diverse Pairwise Microbial Interactions." *eLife* 6.

Mori, Matteo et al. 2016. “Constrained Allocation Flux Balance Analysis” ed. Kiran Raosaheb Patil. *PLoS Computational Biology* 12(6): e1004913.

Nemergut, D. R. et al. 2013. “Patterns and Processes of Microbial Community Assembly.” *Microbiology and Molecular Biology Reviews*.

Orth, Jeffrey D, Ines Thiele, and Bernhard ØPalsson. 2010. “What Is Flux Balance Analysis?” *Nature Biotechnology* 28(3): 245–48.

Parras-Moltó, Marcos, and Daniel Aguirre de Cárcer. 2020. “Detection of Phylogenetic Core Groups in Diverse Microbial Ecosystems.”

Rocha, Inês et al. 2019. “Seed Coating: A Tool for Delivering Beneficial Microbes to Agricultural Crops.” *Frontiers in Plant Science* 10.

Sahraeian, Sayed Mohammad Ebrahim, and Byung-Jun Yoon. 2013. “SMETANA: Accurate and Scalable Algorithm for Probabilistic Alignment of Large-Scale Biological Networks” ed. Peter Csermely. *PLoS ONE* 8(7): e67995.

Schuster, Stefan, and Claus Hilgetag. 1994. “On Elementary Flux Modes in Biochemical Reaction Systems at Steady State.” *Journal of Biological Systems* 02(02): 165–82.

Stein, Richard R et al. 2013. “Ecological Modeling from Time-Series Inference: Insight into Dynamics and Stability of Intestinal Microbiota” ed. Christian von Mering. *PLoS Computational Biology* 9(12): e1003388.

Succurro, Antonella, and Oliver Ebenhöh. 2018. “Review and Perspective on Mathematical Modeling of Microbial Ecosystems.” *Biochemical Society Transactions* 46(2): 403–12.

Wennekes, Paul L, James Rosindell, and Rampal S Etienne. 2012. “The Neutral-Niche Debate: A Philosophical Perspective.” *Acta Biotheoretica* 60(3): 257–71.

Zelezniak, Aleksej et al. 2015. “Metabolic Dependencies Drive Species Co-Occurrence in Diverse Microbial Communities.” *Proceedings of the National Academy of Sciences* 112(20): 6449–54.

Zmora, Niv et al. 2016. “Taking It Personally: Personalized Utilization of the Human Microbiome in Health and Disease.” *Cell Host & Microbe* 19(1): 12–20.

Anexo: *Scripts* desarrollados. Trabajo de Fin de Máster: "Modelización metabólica de comunidades microbianas estables crecidas con fuentes de carbono y energía únicas y simples"

Silvia Talavera Marcos

Scripts desarrollados

Este Anexo recoge los *scripts* desarrollados para automatizar los análisis correspondientes al Trabajo de Fin de Máster "Modelización metabólica de comunidades microbianas estables crecidas con fuentes de carbono y energía únicas y simples" (Silvia Talavera Marcos, Máster en Bioinformática y Biología Computacional, Universidad Autónoma de Madrid, 2020-2021).

A continuación se incluye una breve descripción de todos ellos:

- **modelado.R** incluye el alineamiento con Nucmer, la creación de modelos con CarveMe y (opcionalmente) el análisis con Smetana para una pareja de nodos dada.
- **annotate.R** se encarga de la anotación funcional con eggNOG-mapper y, opcionalmente, la creación de un modelo consenso. También es capaz de llamar a Nucmer para iniciar el proceso desde el principio.

Las funciones definidas para estos scripts se encuentran en **utils.R**.

- **RefrFBA.py** es un *script* que se encarga de ejecutar simulaciones de FBA para todos los modelos de un PCG dado. Devuelve las tasas de crecimiento en forma de archivos *.csv* y por la salida estándar.
- **parser.R** se encarga de generar informes en formato de texto plano que resumen los resultados de Smetana.
- Por último, se han desarrollado dos scripts en Python para crear los consensos: **consenso.py** crea un modelo SBML a partir de otros modelos SBML y **consenso_EGG.py** crea una tabla de anotaciones consenso a partir de múltiples archivos de anotaciones.

Todos los *scripts* aquí incluidos se pueden encontrar también en el repositorio https://github.com/urihs/TFM. En todos los casos se incluye el código del *script* seguido de una explicación del funcionamiento del mismo.

modelado.R

```
#!/usr/bin/env Rscript

start.time <- Sys.time()

# -------------------------
#        FUNCIONES
# -------------------------

library("parallel")
library("optparse")
home <- strsplit(paste0("./",getopt::get_Rscript_filename()),split="/")[[1]]
home <- paste(home[-length(home)],collapse="/")
source(paste0(home,"/utils.R")) # cargo las funciones del paquete



# -------------------------------
# 1 --> Definiciones preliminares
# -------------------------------
```

```
option_list <- list(
  make_option(c("-n", "--nodes"), type="character", default=NULL,
              help="Node input information text file name (e.g. 'my_node_data.txt'). Th
e file needs to have the following format:\n\t\tNode35562    k__Bacteria;p__Proteobacte
ria;c__Gammaproteobacteria;o__Enterobacteriales;f__Enterobacteriaceae;\n\t\tNode27828 k_
_Bacteria;p__Proteobacteria\nThe taxonomy must include at least two fields, with the sec
ond one being the phylum.", metavar="character"),
  make_option(c("-m", "--medium"), type="character", default="M9",
              help="medium (e.g. 'M9', 'M9[glc]'", metavar="character"),
  make_option(c("--mediadb"), type="character", default=paste0(home,"/my_media.tsv"),
              help="media database file name", metavar="character"),
  make_option(c("-c", "--checking"), type="logical", default=FALSE,
              help="TRUE or FALSE (default). If TRUE, generated and analysed models are
limited to those pairs of species that are co-ocurring in pairs in one or more samples o
f a specified experiment.", metavar="logical"),
  make_option(c("-e", "--experiment"), type="character", default=NULL,
              help="Experiment input information text file name (e.g. 'table.from_biom.
tsv'). The tab-separated file needs to have the following format:\n\t\t#OTU ID\tS1\tS2\t
S3\t(...)\n\t\tO1\t000000\t000001\t000002\t(...) ", metavar="character"),
  make_option(c("--coupling"), type="logical", default=TRUE,
              help="If TRUE, smetana computes an aditional analysis without the --no-co
upling option (see smetana help). Default is TRUE.", metavar="logical"),
  make_option(c("--nucmer"), type="character", default=paste0(home,"/MUMmer3.23/nucmer")
,
              help="Path to the nucmer executable (e.g. './MUMmer3.23/nucmer', '~/my_ap
ps/nucmer'.", metavar="character"),
  make_option(c("--showcoords"), type="character", default=paste0(home,"/MUMmer3.23/show
-coords"),
              help="Path to the show-coords executable (e.g. './MUMmer3.23/show-coords'
, '~/my_apps/show_coords'.", metavar="character"),
  make_option(c("--tree"), type="character", default=paste0(home,"/99_otus_nodes.tree"),
              help="16S phylogenetic tree file name. The node names of the trees should
be modified, and a genuine name should be given for all. This could be done for example
using R with the function 'makeNodeLabel' from 'ape' package. Default is './99_otus_node
s.tree' (Greengenes gg_13_5).", metavar="character"),
  make_option(c("--fasta"), type="character", default=paste0(home,"/99_otus.fasta"),
              help="16S sequences to be analyzed in multifasta format. Default is './99
_otus_nodes.tree' (Greengenes gg_13_5).", metavar="character"),
  make_option(c("--db16s"), type="character", default=paste0(home,"/bac120_ssu_reps_r95.
fna"),
              help="16S sequences database. Nucmer will align the tree leaves' 16S sequ
ences to this database. Default is './bac120_ssu_reps_r95.fna' (from GTDB https://data.a
ce.uq.edu.au/public/gtdb/data/releases/release95/95.0/genomic_files_reps/).", metavar="c
haracter"),
  make_option(c("--dbproteins"), type="character", default=paste0(home,"/protein_faa_rep
s/bacteria/"),
              help="Aminoacid sequences database. CarveMe will take files from here tha
t correspond to Nucmer hits and create SBML models from those files. Default is 'protein
_faa_reps/bacteria/' (from GTDB https://data.ace.uq.edu.au/public/gtdb/data/releases/rel
ease95/95.0/genomic_files_reps/).", metavar="character"),
  make_option(c("--run_smetana"), type="logical", default=TRUE,
              help="If TRUE, runs a Smetana analysis for inter-node pairs. If FALSE, sk
ips the Smetana analysis. Default: TRUE.", metavar="logical"),
  make_option(c("--cores"), type="numeric", default=4,
              help="Number of cores to use in parallelization processes (mclapply). Def
ault: 4.", metavar="numeric"))


parser <- OptionParser(option_list=option_list)
```

```
opt <- parse_args(parser)

t <- read.table(opt$nodes,sep = "\t")
nodos      <- levels(t[[1]])
taxonom    <- levels(t[[2]])
medium     <- opt$medium
mediadb    <- opt$mediadb
checking   <- opt$checking
run_smetana<- opt$run_smetana
coupling   <- opt$coupling

if (checking==TRUE & is.null(opt$experiment)) {
  stop("When --checking TRUE, a experiment file must be specified")
  } else if (checking == TRUE) {
  exp = opt$experiment
  }

nucmer_path     <- opt$nucmer
showcoords      <- opt$showcoords

tree_file         <- opt$tree
otus_fasta_file   <- opt$fasta
db_16S            <- opt$db16s
db_protein_folder <- opt$dbproteins

grampospath <- paste0(home,"/grampos.csv")
gramnegpath <- paste0(home,"/gramneg.csv")

cores <- opt$cores

# ------------------------
# 2 --> carga de archivos
# ------------------------
if (!require("ape", quietly=TRUE)) BiocManager::install("ape")
tn = ape::read.tree(tree_file)
nodos_hojas <- mclapply(nodos, function(nodo) {ape::extract.clade(tn, nodo)}, mc.cores=
cores)

# divido el fasta en dos archivos para facilitar su parsing después:
system(paste0("cat ",otus_fasta_file," | grep '>' > 99_otus_col1"))
system(paste0('cat ',otus_fasta_file,' | grep ">" -v  > 99_otus_col2'))

df1 <- read.csv("99_otus_col1",header = F)
fasta <- read.csv("99_otus_col2",header = F)
rownames(fasta) <- sub(">", "", df1[,1]) # elimino ">"
fasta <- as.data.frame(t(fasta)) # Columna 1: > ID. Columna 2: secuencia

system("rm 99_otus_col*") # elimino archivos temporales

# ------------------------------------------------
# 3 --> alinear cada hoja, filtrar y hacer modelos
# ------------------------------------------------
# Asigno secuencias 16S a cada hoja
if (checking == TRUE) {
  # Pares de hojas de los dos nodos. INTER-NODO.
  filtered_pairs <- check(nodos=nodos_hojas, exp=exp,cores=cores)
  # De cada hoja que pase el checking, tomo la secuencia de 16S del fasta original.
  # Son todas las hojas de cada nodo que pasan cualquiera de los dos checkings.
  checked_tipl <- list(levels(filtered_pairs[,1]),levels(filtered_pairs[,2]))
```

```
  nodos_16S    <- mclapply(checked_tipl, function(n) {fasta[n]},mc.cores=cores)
  } else {
    # De cada hoja (sin checking), cojo la secuencia de 16S del fasta original
    nodos_16S    <- mclapply(nodos_hojas, function(nodo) {fasta[nodo$tip.label]},mc.cor
es=cores)
  }


system("mkdir models")
# Para cada nodo creo una carpeta de resultados
for (i in c(1:length(nodos))) {
  filepath   <- paste("models/",nodos[i],"/",sep="")
  system(paste("mkdir", filepath)) # la carpeta tendrá el mismo nombre que el nodo

  # Alineo cada hoja de cada nodo con Nucmer y obtengo un genoma adecuado para cada una,
  # seleccionando solo las hojas cuyos hits pasan un filtro de calidad
  nucmer_res_final <- find_alignment_hits(filepath, nodos_16S[[i]], nucmer_path, db_16S,
showcoords, cores)

  # Modelado con CarveMe de todas las hojas de cada nodo que pasan el filtro de Nucmer
  print(paste0("Creating models for ",nodos[i],"..."))
  dump <- simplify2array(mclapply(nucmer_res_final,
                                  FUN = function(line) {carve(line, taxonom[i], filepa
th, db_protein_folder)},mc.cores=cores))
  print(paste0("Finished modelling for ",nodos[i],"."))

  }

# -------------------------------------
# 4 --> análisis metabólico con Smetana
# -------------------------------------

if (run_smetana == TRUE) {
  # Definimos la lista de parejas a analizar
  if (checking == TRUE) {
    pairs <- filtered_pairs
    } else {
      pairs <- expand.grid(nodos_hojas[[1]]$tip.label, nodos_hojas[[2]]$tip.label, KEEP
.OUT.ATTRS = F)
    }

  # Ejecutamos Smetana para cada pareja inter-nodo de hojas para las que se ha creado
  # un modelo. Esta lista se parejas se guardará en smetana_results/generated_pairs.txt
.
  # También guardamos la lista de parejas que no han sido analizadas por Smetana, en el
  # archivo smetana_results/filtered_out_pairs.txt. Incluimos el porcentaje de parejas
  # que pasan y no pasan el filtro.
  #~~~~~~~~~~~~~~~~~~
  output = "smetana_results/"
  system(paste0("mkdir ",output))
  system(paste0("mkdir ",output,"global"))
  system(paste0("mkdir ",output,"detailed"))
  if (coupling==TRUE) {
    output_coupling = paste0(output,"coupling/")
    system(paste("mkdir",output_coupling))
    system(paste0("mkdir ",output_coupling,"global"))
    system(paste0("mkdir ",output_coupling,"detailed"))
  } else {
    output_coupling = NULL}
```

```
  #~~~~~~~~~~~~~~~~~
  generated_pairs_filename = "generated_pairs.txt"
  dump <- file.create(paste0(output,generated_pairs_filename)) # vaciamos el archivo, de
existir, o lo creamos si no existe

  pairs <- as.data.frame(t(pairs)) # preparamos la matriz para el siguiente paso
  failed_pairs <- mcmapply(pairs, FUN=function(z) {
    smetana(z, modelfilepath = "models/", output=output,
                        coupling=coupling, output_coupling=output_coupling,
                        generated_pairs_filename=generated_pairs_filename)
    }, mc.cores=cores)

  failed_pairs[simplify2array(mclapply(failed_pairs, is.null, mc.cores=cores))] <- NULL

  # Anotamos las parejas que no han sido analizadas por Smetana (no pasaron el filtro de
Nucmer)
  write(x=paste0("filtered out: ", 100*length(failed_pairs)/length(pairs[,1])/2,"%"), f
ile=paste0(output,"filtered_out_pairs.txt"))
  dump <- mclapply(colnames(failed_pairs), FUN=function(col){
    cat(failed_pairs[,col],"\n", file=paste0(output,"filtered_out_pairs.txt"), append=T)
    }, mc.cores=cores)

  print(paste0("Finished SMETANA analysis."))
}

end.time <- Sys.time()
time.taken <- end.time - start.time
print(paste0("Execution time: ",format(time.taken,format = "%H %M %S")))```
```

Uso: `modelado.R [options]`

Argumentos:

Este *script*, como todos los demás, se ejecuta llamándolo desde el terminal. Los argumentos se recogen con la función `OptionParser` de la librería `optparse`. La opción `--help` devuelve una explicación en inglés de todos ellos.

- **`-n, --nodes`**. Nombre del archivo de texto con información sobre los dos nodos de interés (e. g. 'my_node_data.txt'). El formato del archivo debe ser el que se muestra a continuación. La taxonomía debe incluir al menos dos filos, correspondiendo el segundo al filo:

  ```
    Node35562   k__Bacteria;p__Proteobacteria;c__Gammaproteobacteria;o__Enterobacteri
  ales;f__Enterobacteriaceae;

    Node27828   k__Bacteria;p__Proteobacteria
  ```

- **`-m, --medium`**. Medio (e.g. 'M9', 'M9[glc]').
- **`--mediadb`**. Archivo con la base de datos de medios, por defecto "my_media.tsv" incluido en la carpeta original del *script*.
- **`-c, --checking`**. Si `TRUE`, los modelos generados y analizados se limitan a aquellos pares de especies que coinciden en una o más muestra de un experimento especificado (corresponde a la Estrategia 1). `FALSE` (Estrategia 2) por defecto.
- **`-e, --experiment`**. Nombre del archivo de texto con información del experimento de interés (e.g. 'table.from_biom.tsv'). Es un archivo separado por tabulaciones con el siguiente formato:

```
#OTU ID S1  S2  S3  (...)

O1  000000  000001  000002  (...)
```

- **`--coupling`**. Si TRUE, Smetana hace un análisis adicional sin su opción –no-coupling.
- **`--nucmer`**. Ruta del ejecutable de Nucmer (e.g. ‘./MUMmer3.23/nucmer’, ‘~/my_apps/nucmer’).
- **`--showcoords`**. Ruta del ejecutable del show-coords de MUMmer (e.g. ‘./MUMmer3.23/show-coords’, ‘~/my_apps/show_coords’).
- **`--tree`**. Nombre del archivo del árbol filogenético 16S. Los nombres de los nodos deben modificarse para que coincidan con los del archivo de información de nodos (`-n, --nodes`). Esta modificación se puede hacer fácilmente por ejemplo usando la función `makeNodeLabel` del paquete `ape` de R. El árbol por defecto es ‘99_otus_nodes.tree’ (Greengenes gg_13_5), en la carpeta original del *script*.
- **`--fasta`**. Archivo multifasta de secuencias 16S de las hojas del árbol utilizado. El archivo por defecto es’99_otus_nodes.tree’ (Greengenes gg_13_5) en la carpeta original del *script*.
- **`--db16s`**. Base de datos de secuencias 16S contra la cual Nucmer va a alinear las secuencias de las hojas del árbol. Por defecto se usa ‘bac120_ssu_reps_r95.fna’ (GTDB https://data.ace.uq.edu.au/public/gtdb/data/releases/release95/95.0/genomic_files_reps/).
- **`--dbproteins`**. Base de datos de secuencias de aminoácidos. CarveMe tomará de aquí los archivos que correspondan a los *hits* de Nucmer y creará modelos metabólicos a partir de ellos. Por defecto es ‘protein_faa_reps/bacteria/’ (GTDB https://data.ace.uq.edu.au/public/gtdb/data/releases/release95/95.0/genomic_files_reps/).
- **`--run_smetana`**. Si `TRUE`, ejecuta un análisis (global y detallado) de Smetana para parejas internodo. Si `FALSE`, se salta este paso.
- **`--cores`**. Número de núcleos usados en procesos de paralelización (mclapply y mcmapply). Por defecto son 4.

Descripción: `modelado.R` se encarga de generar con CarveMe modelos de los nodos/PCGs especificados en la entrada (`--nodos`) y, opcionalmente, los análisis de Smetana correspondientes.

En primer lugar, si `--checking` es `TRUE`, se seleccionan las hojas de cada nodo que sí coincidían con las hojas del otro nodo en al menos una muestra del experimento especificado (guardado en la variable `exp`). Se hace con la función `check` definida en `utils.R`.

En segundo lugar, se hace un alineamiento con Nucmer de las secuencias 16S (GreenGenes) de las hojas deseadas para cada nodo y una base de datos de genomas completos, GTDB. Solo se seleccionan aquellas asignaciones que superan un filtro fijo (identidad > 97% y cobertura (de la referencia/*hit*) > 90%). A las hojas cuyo alineamiento pase el filtro se les asigna un archivo con una lista completa de todas las proteínas de su genoma. Ese archivo es a partir del cual CarveMe generará modelos SBML. La función principal de este proceso es `find_alignment_hits`.

En tercer lugar, se generan los modelos con CarveMe. Los modelos se guardan con el nombre de la hoja correspondiente. Es uno de los pasos más largos, pero está paralelizado con `mcapply` para ahorrar tiempo de ejecución. Además, si ya existe un modelo con el nombre de esa hoja no se genera de nuevo, permitiendo ahorrar tiempo si la ejecución de `modelado.R` se ha reiniciado. Cada nodo tiene su propia carpeta con el nombre del nodo (el que se indique en el archivo dado con `--nodos`) que está dentro, a su vez, de la carpeta “models”. Esta carpeta se guarda automáticamente en el directorio de trabajo desde el que se ha ejecutado el *script* (que no tiene por qué coincidir con el directorio donde se encuentra el mismo).

Por último, se ejecuta el paso de Smetana, `run_smetana==TRUE`. Primero se crea una lista con todas las parejas inter-nodo posibles de modelos o, si `checking==TRUE`, solo aquellas parejas inter-nodo que coincidieran en la misma muestra del experimento. A continuación hay, para cada pareja, dos o tres

ejecuciones según si `coupling` es `TRUE` o no. En cada ejecución, Smetana va comprobando a) que los modelos SBML existen y b) que el análisis no existiera ya. Asimismo, estas ejecuciones están paralelizadas con `mcapply`. Los análisis se guardan en la carpeta "smetana_results", que también se crea en el directorio de trabajo, con subcarpetas para cada tipo de análisis.

Durante la ejecución se imprimen en la salida estándar mensajes que indican la actividad del *script*. Al final se imprime el tiempo de ejecución.

Salida:

- Modelos SBML para OTUs/hojas de los nodos/PCGs especificados.
- Si `--run_SMETANA` es `TRUE`, análisis globales y detallados de Smetana.
- Si `--coupling` es `TRUE`, se generan también informes detallados sobre los metabolitos acoplados al crecimiento.
- También se generan archivos con el output del alineamiento de Nucmer, incluyendo un archivo `queries_v_hits` para cada PCG.

Requiere: R (3.5.0), CarveMe (1.4.0+), Smetana (1.2.0), NUCmer (cualquier versión; usamos la 3.1; MUMmer 3.23), ape (se instala automáticamente), parallel, optparse

---

annotate.R

```
#!/usr/bin/env Rscript

start.time <- Sys.time() # para devolver al final el tiempo de ejecución
# INPUT: annotate.R -g genomes -o outputname --outputdir outputdir
# INFO: THIS SCRIPT TAKES A LIST OF GENOMES, ANNOTATES ALL OF THEM WITH EGGNOG MAPPER AN
D RETURNS THE ANNOTATED FILES PLUS A CONSENSUS ONE

# =======================
#        FUNCIONES
# =======================
library("optparse")
library("parallel")

home <- strsplit(paste0("./",getopt::get_Rscript_filename()),split="/")[[1]]
home <- paste(home[-length(home)],collapse="/")
source(paste0(home,"/utils.R")) # cargo las funciones del paquete

# ==============
#  Definiciones
# ==============

option_list <- list(
  make_option(c("-g", "--genomes"), type="character", default=NULL,
              help="File containing the list of genomes to annotate. The format may be
the following:\n\tRS_GCF_000281895.1\n\tRS_GCF_900100495.1\n\tRS_GCF_900104015.1\n\t(...
)", metavar="character"),
  make_option(c("-n", "--nodes"), type="character", default=NULL,
              help="Node input information text file name (e.g. 'my_node_data.txt'). Th
e file needs to have the following format:\n\t\tNode35562    k__Bacteria;p__Proteobacte
ria;c__Gammaproteobacteria;o__Enterobacteriales;f__Enterobacteriaceae;\n\t\tNode27828 k_
_Bacteria;p__Proteobacteria\nThe taxonomy must include at least two fields, with the sec
ond one being the phylum.", metavar="character"),
  make_option(c("--skip_consensus"), type="logical", default=FALSE,
              help="If TRUE, eggNOG-mapper creates the annotation files but a consensus
file is not made. Default: FALSE.", metavar="logical"),
  make_option(c("-m", "--medium"), type="character", default="M9",
```

```
              help="medium (e.g. 'M9', 'M9[glc]'", metavar="character"),
  make_option(c("--outputdir"), type="character", default="./annotate_results",
              help="output directory for annotation files and the consensus model(s)",
metavar="character"),
  make_option(c("-o","--outputname"), type="character", default="node_consensus",
              help="output name for the consensus model(s)", metavar="character"),
  make_option(c("--mediadb"), type="character", default=paste0(home,"/my_media.tsv"),
              help="media database file name", metavar="character"),
  make_option(c("-c", "--checking"), type="logical", default=FALSE,
              help="TRUE or FALSE (default). If TRUE, annotated genomes are limited to
those pairs of species that are co-ocurring in pairs in one or more samples of a specifi
ed experiment. Only applicable when input is --nodes.", metavar="logical"),
  make_option(c("-e", "--experiment"), type="character", default=NULL,
              help="Experiment input information text file name (e.g. 'table.from_biom.
tsv'). Only needed when input is --nodes and --checking is TRUE. The tab-separated file
needs to have the following format:\n\t\t#OTU ID\tS1\tS2\tS3\t(...)\n\t\tO1\t000000\t000
001\t000002\t(...) ", metavar="character"),
  make_option(c("--nucmer"), type="character", default=paste0(home,"/MUMmer3.23/nucmer")
,
              help="Path to the Nucmer executable (e.g. './MUMmer3.23/nucmer', '~/my_ap
ps/nucmer'). Only needed when input is --nodes.", metavar="character"),
  make_option(c("--showcoords"), type="character", default=paste0(home,"/MUMmer3.23/show
-coords"),
              help="Path to the show-coords executable (e.g. './MUMmer3.23/show-coords'
, '~/my_apps/show_coords'). Only needed when input is --nodes.", metavar="character"),
  make_option(c("--emapper_path"), type="character", default=paste0(home,"/eggnog-mapper
-master/emapper.py"),
              help="Path to the emapper.py executable file (e.g. './eggnog-mapper-my_ve
rsion/emapper.py').",metavar="character"),
  make_option(c("--tree"), type="character", default=paste0(home,"/99_otus_nodes.tree"),
              help="16S phylogenetic tree file name. Only needed when input is --nodes.
The node names of the trees should be modified, and a genuine name should be given for a
ll. This could be done for example using R with the function 'makeNodeLabel' from 'ape'
package. Default is './99_otus_nodes.tree' (Greengenes gg_13_5).", metavar="character"),
  make_option(c("--fasta"), type="character", default=paste0(home,"/99_otus.fasta"),
              help="16S sequences of all the leaves of the tree, in multifasta format.
Only needed when input is --nodes. Default is './99_otus_nodes.tree' (Greengenes gg_13_5
).", metavar="character"),
  make_option(c("--db16s"), type="character", default=paste0(home,"/bac120_ssu_reps_r95.
fna"),
              help="16S sequences database. Only needed when input is --nodes. Nucmer w
ill align the tree leaves' 16S sequences to this database. Default is './bac120_ssu_reps
_r95.fna' (from GTDB https://data.ace.uq.edu.au/public/gtdb/data/releases/release95/95.0
/genomic_files_reps/).", metavar="character"),
  make_option(c("--dbproteins"), type="character", default=paste0(home,"/protein_faa_rep
s/bacteria/"),
              help="Aminoacid sequences database. EggNOG-mapper will annotate files fro
m here. Default is 'protein_faa_reps/bacteria/' (from GTDB https://data.ace.uq.edu.au/pu
blic/gtdb/data/releases/release95/95.0/genomic_files_reps/).", metavar="character"),
  make_option(c("--dmnd_db"), type="character", default=NULL,
              help="Path to a custom Diamond database. Specially useful when using a Di
amond version different from the eggNOG-mapper one, such as the one created with create_
compatible_database.R.", metavar="character"),
  make_option(c("--cores"), type="numeric", default=4,
              help="Number of cores to use in parallelization processes (mclapply). Def
ault: 4.", metavar="numeric"))

parser <- OptionParser(option_list=option_list, usage = "Desktop/TFM/annotate.R [options
]\n\nThe main input data is either --genomes or --nodes")
```

```
opt <- parse_args(parser)

cores <- opt$cores
skip_consensus <- opt$skip_consensus

if (!is.null(opt$nodes) && is.null(opt$genomes)) {
  skip_alignment <- FALSE
  t <- read.table(opt$nodes,sep = "\t")
  nodos      <- levels(t[[1]])
  taxonom    <- levels(t[[2]])
  nucmer_path<- opt$nucmer
  showcoords <- opt$showcoords
  tree_file         <- opt$tree
  otus_fasta_file   <- opt$fasta
  db_16S            <- opt$db16s

  checking   <- opt$checking
  if (checking==TRUE & is.null(opt$experiment)) {
    stop("When --checking TRUE, a experiment file must be specified")
  } else if (checking == TRUE) {
    exp = opt$experiment
  }

} else if (is.null(opt$nodes) && !is.null(opt$genomes)) {
  skip_alignment=TRUE

} else if (is.null(opt$nodes) && is.null(opt$genomes)) {
  stop("Please provide an input file. It may be either a list of genomes to annotate or
a node data file.")

} else { # both --genomes and --nodes were provided
  stop("Only one kind of input (--genomes or --nodes) can be provided")}

# Annotation variables
outputdir         <- opt$outputdir
outputname        <- opt$outputname
db_protein_folder <- opt$dbproteins
emapper_path      <- opt$emapper_path


if (is.null(opt$dmnd_db)) {
  emapper_folder  <- system(paste0("echo $(dirname ",emapper_path,")"),intern=TRUE)
  dmnd_db         <- paste0(emapper_folder,"/data/eggnog_proteins.dmnd")
} else
  dmnd_db         <- opt$dmnd_db

# ========================
#   Anotar los genomas
# ========================

if (skip_alignment == TRUE) {
  genomes <- scan(genomes,character(),sep="\n")

  #   Anotar el genoma asignado a cada hoja con eggNOG-mapper
  # ------------------------------------------------------------
  annotate(genomes, outputdir, db_protein_folder, emapper_path, cores)
  system(paste0("rm ",outputdir,"/*seed_orthologs"))
```

```
  #   Crear genoma consenso
  # -------------------------
  if (!skip_consensus) {
    system(paste("python3 consenso_EGG.py $(realpath",outputdir,") $(realpath",outputdir
,")",outputname))

  #  Fabricar modelo con CarveMe
  # ----------------------------
    system(paste0("carve --egg ",outputname,".tsv -o ",outputdir,"/",outputname,".xml")
)

  }


} else {

  #   Obtener secuencias 16S para todas las hojas de ambos nodos
  # ------------------------------------------------------------
  if (checking == TRUE) {
    filtered_pairs <- check(nodos=nodos_hojas, exp=exp)
    checked_tipl <- list(levels(filtered_pairs[,1]),levels(filtered_pairs[,2]))
    nodos_16S    <- mclapply(checked_tipl, function(n) {fasta[n]},mc.cores=cores)
  } else {
    nodos_16S    <- mclapply(nodos_hojas, function(nodo) {fasta[nodo$tip.label]},mc.cor
es=cores)
  }

  system("mkdir models")
  # Para cada nodo haremos alineamientos, anotaciones y un consenso

  genomes=list()
  for (i in c(1:length(nodos))) {
    filepath   <- paste("models/",nodos[i],"/",sep="")
    system(paste("mkdir", filepath))

    #    Alineamiento con Nucmer
    # ----------------------------
    nucmer_res_final <- find_alignment_hits(filepath, nodos_16S[[i]], nucmer_path, db_1
6S, showcoords, cores)
    genomes[[i]] <- scan(file=paste0(filepath,"genomes"),what=character(),sep="\n")

    #   Anotar el genoma asignado a cada hoja con eggNOG-mapper
    # ---------------------------------------------------------
    node_outputdir  <- paste0(outputdir,"/",nodos[i])
    node_outputname <- paste0(outputname,"_",nodos[i])
    # En esta(s) carpeta(s) se guardará la lista de los nombres de los genomas obtenido
s por Nucmer
    system(paste0("mkdir ",node_outputdir))

    annotate(genomes[[i]], node_outputdir, db_protein_folder, emapper_path, cores)
    system(paste0("rm ",outputdir,"/*seed_orthologs"))

    if (!skip_consensus) {

    #   Crear genoma consenso
    # -------------------------
      system(paste("python3 consenso_EGG.py $(realpath",node_outputdir,") $(realpath",n
ode_outputdir,")",node_outputname))
```

```
    #  Fabricar modelo con CarveMe
    # ----------------------------
      system(paste0("carve --egg ",node_outputname,".tsv ",gram(taxonom[i])," -o ",node
_outputdir,"/",node_outputname,".xml"))
    }
  }
}


end.time <- Sys.time()
time.taken <- end.time - start.time
print(paste0("Execution time: ",format(time.taken,format = "%H %M %S")))
```

Uso: `annotate.R [options]`

El *input* principal es bien `--nodes` o bien `--genomes`.

Argumentos:

- **`-g, --genomes`**. Ruta de un archivo de texto que contenga una lista de genomas a anotar. El formato debe ser el siguiente, en consonancia con el de la base de datos de genomas de proteínas dada:

  ```
  RS_GCF_000281895.1

  RS_GCF_900100495.1

  RS_GCF_900104015.1

  (...)
  ```

- **`-n, --nodes`**. Ruta de un archivo de texto con información de los nodos de interés. El formato debe ser el siguiente:

  ```
  Node35562   k__Bacteria;p__Proteobacteria;c__Gammaproteobacteria;o__Enterobacteri
ales;f__Enterobacteriaceae;

  Node27828   k__Bacteria;p__Proteobacteria
  ```

- **`--skip_consensus`**. Si `TRUE`, eggNOG-mapper crea los archivos anotados pero no se crean modelos ni un consenso a partir de los mismos. Por defecto es `FALSE`.
- **`-m, --medium`**. Medio (e. g. 'M9', ' M9[glc]').
- **`--outputdir`**. Directorio de salida para los archivos de anotación y los modelos consenso.
- **`-o, --outputname`**. Nombre del modelo consenso creado (si el *input* es `-g`) o nombre común de los modelos consenso (si el *input* es `-n` y son varios nodos).
- **`--mediadb`**. Ruta del archivo de medios de cultivo.
- **`--checking`**. Si `TRUE`, solo se anotan aquellos genomas cuyas secuencias hayan sido detectadas en el experimento especificado. Solo se aplica si se hace alineamiento.
- **`-e, --experiment`**. Ruta del archivo con información del experimento de interés cuando `--checking` es `TRUE` (e. g. 'table.from_biom.tsv'). Solo es necesario si se hace alineamiento y `--checking` es `TRUE`. El archivo debe tener el siguiente formato:

  ```
  #OTU ID S1   S2   S3   (...)
  O1   000000   000001   000002   (...)
  ```

- `--nucmer`. Ruta del ejecutable de Nucmer (e. g. ‘./MUMmer3.23/nucmer’, ‘~/my_apps/nucmer’). Solo es necesario si se hace alineamiento.
- `--show-coords`. Ruta del ejecutable de show-coords (e. g. ‘./MUMmer3.23/show-coords’, ‘~/my_apps/show_coords’). Solo es necesario si se hace alineamiento.
- `--emapper_path`. Ruta del ejecutable de eggNOG-mapper (e. g. ‘./eggnog-mapper-my_version/emapper.py’).
- `--tree`. Nombre del archivo del árbol filogenético 16S. Solo es necesario si se hace alineamiento. Los nombres de los nodos deben modificarse para que coincidan con los del archivo de información de nodos (`-n, --nodes`). Esta modificación se puede hacer fácilmente por ejemplo usando la función `makeNodeLabel` del paquete `ape` de R. El árbol por defecto es ‘99_otus_nodes.tree’ (Greengenes gg_13_5), en la carpeta original del *script*.
- `--fasta`. Archivo multifasta de secuencias 16S de las hojas del árbol utilizado. Solo es necesario si se hace alineamiento. El archivo por defecto es’99_otus_nodes.tree’ (Greengenes gg_13_5) en la carpeta original del *script*.
- `--db16s`. Base de datos de secuencias 16S contra la cual Nucmer va a alinear las secuencias de las hojas del árbol. Solo es necesario si se hace alineamiento. Por defecto se usa ‘bac120_ssu_reps_r95.fna’ (GTDB https://data.ace.uq.edu.au/public/gtdb/data/releases/release95/95.0/genomic_files_reps/).
- `--dbproteins`. Base de datos de secuencias de aminoácidos. EggNOG-mapper tomará de aquí los genomas a anotar, ya sean dados por los *hits* de Nucmer (–nodes) o por el usuario (–genomes). Por defecto es ‘protein_faa_reps/bacteria/’ en la carpeta original del *script* (GTDB https://data.ace.uq.edu.au/public/gtdb/data/releases/release95/95.0/genomic_files_reps/).
- `--dmnd_db`. Ruta de una base de datos de Diamond personalizada. Especialmente útil cuando se utiliza una base de datos diferente a la estándar de eggNOG-mapper, como puede ser una creada con create_compatible_database.R.
- `--cores`. Número de cores a usar en la paralelización de procesos (`mclapply`). Por defecto son 4.

Descripción: En primer lugar se obtiene la lista de genomas a anotar. Esta lista puede tomarse directamente de los datos de entrada (archivo de texto dado con la opción `--genomes`) o bien realizando un alineamiento del nodo o los nodos especificados con `--nodos`. El proceso en este caso es similar al explicado para `modelado.R`, utilizando la función `find_alignment_hits`.

En segundo lugar, cada lista de genomas es anotada con eggNOG-mapper mediante la función `annotate` definida en `utils.R`. Este proceso está paralelizado internamente con `mcapply`.

Por último, si se activó la opción de crear consenso, se hace una llamada a `consenso_EGG.py` y finalmente a CarveMe.

Salida: Archivos de anotaciones de genomas para cada genoma de interés. Si `--skip_consensus` es `FALSE`, también se genera un modelo SBML-FCB2 consenso para cada nodo.

Requiere: R (3.5.0), CarveMe (1.4.0+), eggNOG-mapper (2.0.1), parallel, optparse. Si se utiliza la función de alineamiento con Nucmer, se requieren también NUCmer (cualquier versión; usamos la 3.1; MUMmer 3.23) y ape (se instala automáticamente).

---

create_compatible_database.R

```
#!/usr/bin/env Rscript

# ------------------------------------------------------------
# Crear una base de datos compatible con el diamond del PATH
# ------------------------------------------------------------
library("optparse")

option_list <- list(
  make_option(c("-o","--outputdir"), type="character", default=".",
              help="Output directory for the new database.", metavar="character"),
  make_option(c("-e","--emapper_path"), type="character", default=NULL,
              help="EggNOG-mapper directory. Contains the old database and the old Diam
ond version.", metavar="character"),
  make_option(c("-d","--diamond"), type="character", default="diamond",
              help="New Diamond version location. Default is 'diamond' (PATH).", metava
r="character"))

parser <- OptionParser(option_list=option_list)
opt <- parse_args(parser)

outputdir <- opt$outputdir
diamond   <- opt$diamond

if (is.null(opt$emapper_path)) {
  stop("Please specify the eggNOG-mapper path")
} else {
  emapper_path <- opt$emapper_path
}
old_db       <- paste0(emapper_path,"/data/eggnog_proteins.dmnd")
old_dmnd     <- paste0(emapper_path,"/bin/diamond")

make_compatible_database <- function(emapper_path) {
  print(paste0(old_dmnd," getseq --db ",old_db," > '",outputdir,"/eggnog_proteins.faa';
               ",diamond," makedb --in '",outputdir,"/eggnog_proteins.faa' --db ",output
dir,"/eggnog_proteins_compatible"))
}

make_compatible_database(emapper_path)

print(paste0("New Diamond database saved as: ",outputdir,"/eggnog_proteins_compatible.dm
nd"))
```

Al utilizar eggNOG-mapper junto a otra herramienta que use Diamond (como CarveMe), o simplemente por tener otra versión de Diamond instalada, es posible que se importe al entorno una versión de Diamond diferente a la que viene con eggNOG-mapper. En ese caso, podría haber incompatibilidad entre dicha versión de Diamond y la base de datos de Diamond incluida con el ejecutable de eggNOG-mapper (/data/eggnog_proteins.dmnd) y consecuentemente un fallo a la hora de anotar.

Este breve *script* sirve para obtener una base de datos Diamond en un formato distinto a la utilizada por eggNOG-mapper y compatible con la versión de Diamond que se esté utilizando o se quiera utilizar. Posteriormente se puede indicar a `annotate.R` que utilice esta nueva base de datos mediante `--dmnd_db`.

Uso: `create_compatible_databases.R [options]`

Argumentos:

- **`-o, --outputdir`**. Ruta del directorio de salida para la nueva base de datos.
- **`-e, --emapper_path`**. Ruta de la versión de eggNOG-mapper (contiene la versión antigua de

Diamond y su base de datos antigua).

- `-d, --diamond`. Versión de Diamond para la nueva base de datos. Si no se especifica, se usa la versión del PATH.

Descripción: Este *script* toma los datos de entrada y ejecuta un primer comando que convierte la base de datos antigua, en formato *.dmnd*, en un archivo *.faa*.

El segundo y último comando que ejecuta llama a la nueva versión de Diamond para crear, a partir del archivo .faa, una nueva base de datos en formato .dmnd. El archivo .faa se guarda también en la carpeta de salida.

Salida: /eggnog_proteins.faa y /eggnog_proteins_compatible.dmnd

Requiere: eggNOG-mapper (cualquier versión), Diamond (cualquier versión)

---

utils.R

```
#!/usr/bin/env Rscript
library(parallel)

nucmer <- function(nucmer_path, db_16S, fasta_16S) {
  # Alinea cada secuencia del archivo fasta indicado ("fasta_16S") con las secuencias
  # de la base de datos ("db_16S") y devuelve todos los hits en un archivo llamado
  # "./out.delta".
  system(paste(nucmer_path, db_16S, fasta_16S))
}




carve = function(line, taxonom, outputpath, db_protein_folder) {
  # Dado un string del tipo "<ID de hoja> <ID de genoma anotado>", crea un
  # modelo metabólico de dicha hoja a partir del archivo correspondiente al
  # genoma dado.
  leaf = strsplit(line, split=" ")[[1]][1]
  file = strsplit(line, split=" ")[[1]][2]
  outf = paste0(outputpath,leaf,".xml")
  if (file.exists(outf)) {
    print(paste0(outf," model already exists. Moving to the next one..."))
  } else {
    system(paste0("carve ",db_protein_folder,file,"_protein.faa ",
                  gram(taxonom)," -o ",outf)) # nombres de archivo = nombre de hoja
    print(paste0(outf," model created"))
  }
}




smetana = function(pair, modelfilepath="models/", output, coupling=TRUE, output_coupling
=NULL, generated_pairs_filename="generated_pairs.txt") {
  # Dado un string del tipo <hoja1> <hoja2>, lanza smetana para las hojas dadas,
  # si existen sus archivos. Si no existe alguno de los dos archivos o ninguno,
  # devuelve sus nombres. Si coupling==TRUE, se computan también los análisis sin
  # la opción de smetana "--no-coupling".
  # File creation
  if (coupling==TRUE & is.null(output_coupling)) {
    stop("A output name for the coupling results must be specified.")
  }
```

```
  filepath1 = paste0(modelfilepath,nodos[1],"/")
  filepath2 = paste0(modelfilepath,nodos[2],"/")
  m1 = pair[[1]]
  m2 = pair[[2]]
  if (file_test("-f",paste0(filepath1,m1,".xml")) & file_test("-f",paste0(filepath2,m2,"
.xml"))) {
    for (i in c("global","detailed")){
      output_filename=paste0(output,i,"/",m1,"_",m2,"_",medium)
      if (!file.exists(paste0(output_filename,"_",i,".tsv"))) { #solo crea el archivo s
i no existía ya
        print(paste0("Creating report ",output_filename,"_",i,"..."))
        system(paste0("smetana --",i," ",filepath1,m1,".xml"," ",filepath2,m2,".xml","
--flavor bigg -m ",medium,
                      " --mediadb ",mediadb," --molweight --no-coupling -o ",output_fil
ename))
        write(paste(m1,m2),file=paste0(output, generated_pairs_filename),append=TRUE) #
lista de parejas que se han analizado con Smetana
      } else {
        print(paste0(output_filename,"_",i,".tsv already exists. Moving to the next pai
r..."))
      }
    }
    if (coupling==TRUE) {  # se hace una ejecución más, pero solo para detailed.
      output_filename_c=paste0(output_coupling,i,"/",m1,"_",m2,"_",medium)
      if (!file.exists(paste0(output_filename,"_",i,".tsv"))) { #solo crea el archivo s
i no existía ya
        print(paste0("Creating report ",output_filename,"_",i,"..."))
        system(paste0("smetana --detailed"," ",filepath1,m1,".xml"," ",filepath2,m2,".x
ml"," --flavor bigg -m ",medium,
                      " --mediadb ",mediadb," --molweight -o ",output_filename_c))
      } else {
        print(paste0(output_filename_c,"_detailed.tsv already exists. Moving to the nex
t pair..."))
      }
    }
  } else {
    return(pair) # se devuelve
  }
}




emapper = function(input_fa, db_protein_folder, outputname, outputdir, emapper_path, cor
es) {
  system(paste0(emapper_path," -m diamond --cpu ",cores," --no_annot --no_file_comments
-i ",
                db_protein_folder, input_fa,"_protein.faa"," -o ",outputname," --output
_dir ",
                outputdir," --temp_dir /dev/shm --dmnd_db $PWD/",dmnd_db," --override")
)
  returned <- system(paste0(emapper_path," --annotate_hits_table ",outputdir,"/",outputn
ame,
                            ".emapper.seed_orthologs -o ",outputname," --output_dir ",o
utputdir," --override"))
  if (returned != 0) {
    stop("eggNOG-mapper returned non-zero status. If your Diamond version is different
from the eggNOG-mapper one
         (e.g. the CarveMe version is in the PATH) please create a new database with
```

```
        create_compatible_database.R
        and select it with --dmnd_db when next running annotate.R")
  }
}



check = function(nodos, exp, cores=4) {
  # Dada una pareja de nodos ("nodos", formato ape::phylo) y la ruta de un archivo con
  # datos experimentales ("exp"), determina y devuelve las combinaciones de OTUs
  # inter-nodo que coinciden en una misma muestra experimental
  exp = read.csv(exp,sep="\t",skip = 1,row.names=1)

  # Seleccionamos solo las hojas que estaban en el experimento, para agilizar el proceso
  lista_nodos <- mclapply(nodos, function(nodo) {nodo$tip.label[nodo$tip.label %in% rown
ames(exp)]},mc.cores=cores)

  # Obtengo todas las combinaciones de hojas para cada nodo
  pairs = expand.grid(lista_nodos[[1]], lista_nodos[[2]], KEEP.OUT.ATTRS = F)

  # Selecciono las parejas presentes en una misma muestra
  pairs[,3] = vector(mode="logical",length=length(pairs[,2])) # indicador de si coincid
en o no
  for (muestra in colnames(exp)){
    # anoto qué otus hay en cada muestra
    otus=rownames(subset(exp[muestra],exp[muestra]!=0))
    # indico para cada pareja si coinciden en esa muestra o en otra ya comprobada
    pairs[,3] = pairs[,3] | pairs[,1] %in% otus & pairs[,2] %in% otus
  }
  filtered_pairs  = subset(pairs,pairs[,3]==TRUE)[,0:2]
  return(filtered_pairs)
}



gram  = function(taxonom, gramneg = "gramneg.csv", grampos = "grampos.csv") {
  # Dado un string con la taxonomía en formato GreenGenes, devuelve si es
  # gram-positiva o gram-negativa. Si no se reconoce como G+ ni G-, devuelve
  # un string vacío.
  phylum  <- strsplit(taxonom, split=";")[[1]][2]
  gramneg <- levels(read.table(gramnegpath)[[1]])
  grampos <- levels(read.table(grampospath)[[1]])

  if (phylum %in% gramneg) {
    result <- "-u gramneg"
  }
  else if (phylum %in% grampos) {
    result <- "-u grampos"
  }
  else {
    result <- ""
  }
  return(result)
}
```

```
find_alignment_hits = function(filepath, node_16S, nucmer_path, db_16S, showcoords, core
s) {
  # Runs nucmer on each of the leaves of a given node (input is 16S sequences).
  # Creates 4 different files:

  #    - nucmer_unfiltered: table including all the best hits for every leaf before the
  #                         97% identity and 90% coverage filter.

  #    - nucmer_filtered: table including the best hits with over 97% identity and 90%
  #                       coverage. Some leaves may be filtered out at this point.
  #    - queries_v_hits: list of every leaf that passed the filter vs its best database
  #                      hit according to Nucmer. It's a two-column table

  #    - genomes: list of every database hit that passed the filter. One-column table.
  #

  # Returns a character vector with the leaf names and hits (the same list of the
  # queries_v_hits file).


  # Creo un archivo temporal en formato fasta que contenga las secuencias de ese nodo
  write(
    simplify2array(mclapply(colnames(node_16S), FUN=function(hoja){
      paste(paste(">",hoja,sep=""),as.character(node_16S[hoja][,]),sep="\n")},mc.cores=c
ores)),
    file="sec_temp.fasta")

  # Alineo cada una de sus hojas con la base de datos
  nucmer(nucmer_path, db_16S, "sec_temp.fasta")

  system(paste("mv out.delta",filepath))

  # Descarto hits con identidad por debajo de 97% con show-coords
  nucmer_res <- system(paste(showcoords," -c -l -I 97 ",filepath,
                             "out.delta",sep=""), intern = TRUE)
  # -c  Include percent coverage information
  # -l  Include the sequence length information
  # -I float    Set minimum percent identity to display

  # Creo archivo de mejores resultados sin filtrar por coverage
  header = nucmer_res[4:5]
  nucmer_res = gsub("|", "", nucmer_res[-(0:5)], fixed=T) # elimino separador
  nucmer_res = gsub("(?<=[\\s])\\s*|^\\s+|\\s+$", "", nucmer_res, perl=TRUE) # fusiono
espacios

  write(header, paste(filepath,"nucmer_unfiltered",sep=""))
  write(nucmer_res, paste(filepath,"nucmer_unfiltered",sep=""),append = TRUE) # guardo

  # Creo archivo de mejores resultados filtrados por %ID y coverage y lo guardo
  write(header, paste(filepath,"nucmer_filtered",sep=""))

  system(paste("awk '{ if ($10>=90) { print } }' ", # filtro por cobertura
               filepath,"nucmer_unfiltered > ",
               filepath,"nucmer_temp1",sep=""))
  system(paste("sort -r -k13,13 -k7,7 ", # ordeno por query y por %ID
               filepath,"nucmer_temp1 > ",
               filepath,"nucmer_temp2",sep=""))
```

```
  system(paste("awk -F ' ' -v q='' '{if ($13!=q) {q=$13; print}}' ", # escojo los mejore
s resultados
               filepath,"nucmer_temp2 >>",
               filepath,"nucmer_filtered",sep="")) #guardo

  # Lista filtrada de hits a modelar
  queries_v_hits = system(paste("awk -F ' ' -v q='' '{if ($13!=q) {q=$13; print $13,$12}
}' ",
                                filepath,"nucmer_temp2",sep=""), intern = TRUE)

  # guardo parejas como archivo (a modo de índice informativo)
  write(queries_v_hits, file=paste0(filepath, "queries_v_hits"))

  # guardo solo los genomas en un archivo (útil para anotar posteriormente por ejemplo)
  system(paste("awk -F ' ' -v q='' '{if ($13!=q) {q=$13; print $12}}' ",
               filepath,"nucmer_temp2 > ",filepath,"genomes",sep=""), intern = TRUE)

  # Borrado de archivos temporales
  system(paste("rm sec_temp.fasta ", filepath,"nucmer_temp*",sep=""))

  return(queries_v_hits)
}




annotate = function(genomes, outputdir, db_protein_folder, emapper_path, cores) {
  if (!file.exists(outputdir)){
    system(paste0("mkdir ",outputdir)) }

  N = length(genomes) # número total de genomas
  dump <- simplify2array(mclapply(genomes,FUN=function(genome) {
    print(paste0("Annotating genome ",genome," (total: ",N,")"))
    emapper(input_fa=genome,
            db_protein_folder = db_protein_folder,
            outputname=genome,
            outputdir=outputdir,
            emapper_path=emapper_path,
            cores=cores)
  },mc.cores=cores))
}
```

Uso: -

Argumentos: -

Descripción: Este *script* recoge las funciones más importantes de `modelado.R` y `annotate.R`. `nucmer`, `carve`, `emapper` y `smetana` lanzan estos 4 programas desde el terminal, preparando el comando a lanzar en cada caso e incluyendo en ocasiones mensajes de error si falta algún argumento. `annotate` es una función sencilla que se encarga de llamar a `emapper` para cada genoma.

Las funciones `carve` y `smetana` son un poco más complejas. En primer lugar, dado que deben ejecutar cientos o incluso miles de comandos según el caso (uno por cada modelo o análisis generado), los comandos están paralelizados con `mclapply` y `mcmapply`. Además, en el caso de `carve`, antes de ejecutar cada comando se comprueba si ya existe un modelo con ese nombre (generado en alguna ejecución previa de `modelado.R`), para evitar dedicar tiempo computacional a generar el mismo modelo dos o más veces. En el caso de `smetana` se comprueba antes de cada comando que no se haya generado anteriormente un análisis con el mismo nombre, por la misma razón.

`smetana` se encarga además de implementar o no un tercer análisis, el de los metabolitos acoplados al crecimiento, dependiendo de si la opción de `coupling` está activada.

La función `gram` es llamada desde `carve`. Esta función se encarga de, dado un filo taxonómico, devolver si es Gram-positivo o Gram-negativo. Así, `carve` utilizará el modelo universal más adecuado en cada caso. Se basa en dos archivos que contienen breves listas de filos Gram-positivos y Gram-negativos y que se encuentran en la carpeta original del *script*.

`check` es la función encargada de hacer la selección de modelos para la Estrategia 1. En primer lugar obtiene todas las posibles combinaciones de parejas entre el par de nodos dado. A continuación, recorre con un bucle todas las muestras del archivo del experimento dado y selecciona aquellas parejas que coinciden en al menos una.

`find_alignment_hits` es la función principal encargada del alineamiento, llamada en `modelado.R` y en `annotate.R`. Lo que hace es ejecutar `nucmer` para cada una de las hojas de cada nodo dado y aplica el filtro de calidad de identidad > 97% y cobertura (de la referencia/*hit*) > 90%. Devuelve un vector con el nombre de cada hoja (cuyo alineamiento haya superado el filtro de calidad) y su correspondiente *hit*. Este vector es guardado a modo de tabla en un archivo, “queries_v_hits”. También se generan los archivos “nucmer_unfiltered” y “nucmer_filtered”, con la salida de Nucmer antes y después del filtro de cobertura, y el archivo “genomes”, que consiste en una lista de todos los *hits* que han pasado el filtro. Este último es el tipo de archivo que se da como entrada a `annotate.R` usando `--genomes`.

Salida: -

Requiere: R (3.5.0), optparse, parallel. `nucmer` requiere el programa Nucmer (siendo válida cualquier versión; usamos la 3.1; MUMmer 3.23). `smetana` requiere Smetana (1.2.0). `carve` requiere CarveMe (1.4.0+). `emapper` requiere eggNOG-mapper (2.0.1).

RefrFBA.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec  4 12:24:06 2020

@author: Silvia Talavera Marcos
"""

from reframed import load_cbmodel
import os, sys
from reframed import FBA, Environment
from carveme.reconstruction.utils import load_media_db
import statistics as stats
import pandas as pd

# INPUT: RefrFBA.py input_folder/wd mediadb medium outputdir outputname

# IMPRIME la media y la desviación típica del crecimiento de todos los
#         modelos de la carpeta dada como entrada
# GENERA un archivo .csv con todas las tasas de crecimiento

wd = sys.argv[1]
mediadb = sys.argv[2]
medium = sys.argv[3]

if len(sys.argv) > 4:
    outputdir = sys.argv[4].rstrip("/") # si el usuario da el outputdir
```

```
    if len(sys.argv) > 5:
        outputname = sys.argv[5] # si el usuario da el nombre del output
    else:
        outputname = "report_reframed"+wd.split("/")[-1]+".csv"
else:
    outputdir = wd


models = []
for sbml in os.listdir(wd):
  if ".xml" in sbml:
    models.append(wd+"/"+sbml)

media_db = load_media_db(mediadb, compound_col="compound")

results = [] # lista de listas que finalmente dará un dataframe/csv con todos los datos

init_env = Environment.from_compounds(media_db[medium])
print("\nMedio: "+medium+"\n==============")
total=[]
n=0# para hacer la media de solution.fobj
for f in models:
    model = load_cbmodel(f,flavor="fbc2")
    init_env.apply(model)
    solution = FBA(model,   objective="Growth", get_values=False)
    if solution.fobj>0:
        n+=1
        total.append(solution.fobj)
    results.append([medium,f.split("/")[-1],solution.fobj])
try:
    media = sum(total)/n
    print("media del nodo: ", media)
    print("desviacion típica: ", stats.stdev(total),"\n")
except:
    pass

all_data=pd.DataFrame(results,columns=["Medio","Modelo","Crecimiento"])
all_data.to_csv(outputdir+"/"+outputname,index=False)
```

Uso: RefrFBA.py input_folder mediadb medium outputdir outputname

Los argumentos se indican sin *flags* en este *script*.

Argumentos:

- **`input_folder`**. Una carpeta que contenga todos los modelos SBML-FBC2 que se quiera analizar. Puede ser, por ejemplo, una de las carpetas generadas con `modelado.R`.
- **`mediadb`**. Base de datos de medios de cultivo a utilizar.
- **`medium`**. Medio en el que simular el crecimiento de los modelos dados.
- **`outputdir`**. Directorio de salida para el informe generado.
- **`outputname`**. Nombre del informe generado (incluyendo extensión *.csv*).

Descripción: En primer lugar se hacen los preparativos: a partir de la carpeta dada como `input_folder`, se crea una lista que contiene todos los modelos, filtrando por extensión *.xml*. También se carga el medio de cultivo y se inicializa como “entorno” con el método clase `Environment.from_compounds`, importado de CarveMe.

A continuación, se inicia un bucle en el que cada modelo de la lista será cargado con la función de CarveMe

`load_cbmodel`, se le aplicará el medio de cultivo y se incorporará a una simulación FBA con la función de ReFramed `FBA`. De cada simulación se guarda la tasa de crecimiento obtenida y, si es mayor que 0, se incorporará al cálculo de la tasa media de crecimiento para el total de los modelos. Todas las tasas de crecimiento se guardan en una lista de listas, `results`.

Por último, se imprime por la salida la tasa media de crecimiento y la desviación típica. Los datos de `results` se convierten a `DataFrame` de `pandas` y se guardan como *.csv* con el nombre indicado.

Salida: Archivo *.csv* con todas las tasas de crecimiento junto al nombre de cada modelo. Salida estándar: tasa media de crecimiento y desviación típica.

Requiere: Python (3.6+; en este Trabajo se usa 3.6.9). CarveMe (1.4.0+; incluye las funciones importadas, Pandas y ReFramed).

RefrFBA_E1.py

El *script* `RefrFBA.py` tiene una versión alternativa, `RefrFBA_E1.py`, que recibe como input una carpeta ligeramente distinta, con la siguiente estructura:

```
input_folder
├── cit
│   ├── Node28866
│   │   ├── 4370747.xml
│   │   └── (...)
│   └── Node35562
│       ├── 3944484.xml
│       └── (...)
├── glc
│   ├── Node27828
│   │   └── (...)
│   └── Node35562
│       └── (...)
└── leu
    ├── Node13821
    │   └── (...)
    └── Node28853
        └── (...)
```

`RefrFBA_E1` funciona de la misma manera, solo que el bucle va cambiando el medio especificado automáticamente para cada subcarpeta (M9[*cit*], M9[*glc*] o M9[*leu*]). El *.csv* final incluye todos los datos juntos, especificando el medio de cultivo y añadiendo también una columna con la tasa de crecimimento media. Se incluye el código en el desplegable a continuación.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec  4 12:24:06 2020

@author: Silvia Talavera Marcos
"""

from reframed import load_cbmodel
import os
from reframed import FBA, Environment
from carveme.reconstruction.utils import load_media_db
import statistics as stats
import pandas as pd

wd = "/home/urihs/Desktop/TFM_private/08_reframed/sin_gapfill/"
```

```
models = {}
for medium in os.listdir(wd):
    models[medium] = {}
    for node in os.listdir(wd+medium):
        if "Node" in node:
            models[medium][node] = {}
        else:
            continue
        for sbml in os.listdir(wd+medium+"/"+node):
            if ".xml" in sbml:
                models[medium][node][sbml]=wd+medium+"/"+node+"/"+sbml

media_db = load_media_db("/home/urihs/Desktop/TFM_private/08_reframed/test_media.txt", c
ompound_col="compound")

results = [] # lista de listas que finalmente dará un dataframe/csv con todos los datos
for medium in models.keys():
    medio = "M9["+medium+"]"
    # medio = "LB"
    init_env = Environment.from_compounds(media_db[medio])
    print("\nMedio: "+medio+"\nNodos de: "+medium+"\n==============")
    media = 0
    listofkeys = list(models[medium].keys())
    listofkeys.sort()
    for node in listofkeys:
        media_prev = 0+media
        print("Nodo: "+node+"\n===============")
        total=[]
        n=0# para hacer la media de solution.fobj
        for f in models[medium][node]:
            model = load_cbmodel(models[medium][node][f],flavor="fbc2")
            init_env.apply(model)
            solution = FBA(model,   objective="Growth", get_values=False)
            print(solution)
            if solution.fobj!=0:
                n+=1
                total.append(solution.fobj)
            results.append([medium,node,f,solution.fobj])
        try:
            media = sum(total)/n
            print("media del nodo: ", media)
            print("desviacion típica: ", stats.stdev(total),"\n")
        except:
            pass
    try:
        print("RATIO : ",media_prev/media)
    except:
        pass

all_data=pd.DataFrame(results,columns=["Medio","Nodo","Modelo","Crecimiento"])
all_data.to_csv("report_reframed.csv",index=False)
```

parser.R

```
#!/usr/bin/env Rscript

library("optparse")
```

```
library("parallel")

# ==================
#     Take input
# ==================
### HELP ###
# parser.R -d/-g input_file -c cores --report_name reportname(can include path)

option_list <- list(
  make_option(c("-g", "--global"), type="character", default=NULL,
              help="Input folder with Smetana global results (e.g. ./my_results/NodeXXX
XX/smetana_results/global).", metavar="character"),
  make_option(c("-d", "--detailed"), type="character", default=NULL,
              help="Input folder with Smetana global results (e.g. ./my_results/NodeXXX
XX/smetana_results/global).", metavar="character"),
  make_option(c("-c","--cores"), type="numeric", default=4,
              help="Number of cores to use in parallelization", metavar = "character"),
  make_option(c("--report_name"), type="character", default="report.txt",
              help="Name of the output report file. Default: ./report.txt", metavar="ch
aracter")
)

parser <- OptionParser(option_list=option_list)
opt <- parse_args(parser)

global      <- opt$global
detailed    <- opt$detailed
cores       <- opt$cores
report_name <- opt$report_name

if (is.null(global)) {
  if (is.null(detailed)) {
    stop("Please specify an input folder with --global or --detailed.")
  } else {
    is.detailed <- TRUE
    is.global   <- FALSE
  }
} else if (is.null(detailed)) {
  is.global   <- TRUE
  is.detailed <- FALSE
} else {
  stop("Only --global or --detailed input is accepted. Please choose only one.")
}

if (is.global){
  # Open the dataset
  # ================
  filenames  <- list.files(global, pattern="*.tsv", full.names=TRUE)

  dataset    <- do.call("rbind", mclapply(filenames, FUN=function(filename)  {
      df <- read.table(filename, sep="\t", header=TRUE, na.strings = "n/a")
      rownames(df) <- tail(strsplit(filename,split="/")[[1]],n=1)
      return(df)
    } ,mc.cores=cores))


  # Report MRO=n/a files
  # ====================
  mro.is.na <- as.logical(is.na(dataset["mro"]))
```

```
  row_names  <- filenames
  not_growing <- row_names[mro.is.na]


  #  MRO values
  # =============
  growing <- dataset[!mro.is.na,]

  if (length(growing)==0) {
    write("MRO was n/a for all files. The co-culture can't grow.",file=report_name)
    stop ("MRO was n/a for all files. The co-culture can't grow.")
  } else {
    mean_mro <- mean(growing[,"mro"])
    min_mro  <-  min(growing[,"mro"])
    max_mro  <-  max(growing[,"mro"])
  }

  # MIP report
  # ==========
  mip.is.na <- as.logical(is.na(growing[,"mip"]))
  mip.is.not.na <- growing[!mip.is.na,]

  mean_mip <- mean(mip.is.not.na[,"mip"])
  min_mip  <-  min(mip.is.not.na[,"mip"])
  max_mip  <-  max(mip.is.not.na[,"mip"])


  # Print report
  # ============
  write("The following files have n/a MRO and MIP, which means they can't grow by themse
lves:\n",file=report_name)
  write(not_growing,file=report_name,append=TRUE)

  write("\n\nMRO (metabolic resource overlap) calculates how much the species compete fo
r the same metabolites", file=report_name, append=TRUE)
  write(paste("\nMean MRO:",   mean_mro, sep="\n"), file=report_name, append=TRUE)
  write(paste("\nMinimum MRO:", min_mro, sep="\n"), file=report_name, append=TRUE)
  write(paste("\nMaximum MRO:", max_mro, sep="\n"), file=report_name, append=TRUE)

  write("\n\nMIP (metabolic interaction potential) calculates how many metabolites the s
pecies can share to decrease their dependency on external resources", file=report_name,
append=TRUE)
  write(paste("\nMean MIP:",   mean_mip, sep="\n"), file=report_name, append=TRUE)
  write(paste("\nMinimum MIP:", min_mip, sep="\n"), file=report_name, append=TRUE)
  write(paste("\nMaximum MIP:", max_mip, sep="\n"), file=report_name, append=TRUE)

  write("\n\nFiles where MRO values are available:", file=report_name, append=TRUE)
  write.table(growing, file=report_name, sep="\t", quote=FALSE, append=TRUE)


} else {
  # Open the data files
  # ===================
  files_names   <- list.files(detailed, pattern="*.tsv", full.names=TRUE)

  files_text  <- mclapply(files_names, FUN=function(filename)  {
    read.csv(filename, sep="\t", header=TRUE)
  } ,mc.cores=cores)
```

```
  names(files_text) <- files_names


  # Report empty files
  # ==================
  is.empty  <- as.numeric(mclapply(files_text,FUN=function(x){length(x[,1])},mc.cores=co
res))==0

  empty     <- files_names[is.empty]
  not.empty <- files_names[!is.empty]


  # Analyze metabolites
  # ===================
  merged.files    <- do.call("rbind", files_text)
  rm(files_text) # free memory


  # Create node columns
  check_node      <- function(x,node_1_index,tags=c("first","second")){
    if (x %in% node_1_index) {result <- tags[1]} else {result <- tags[2]}
    return(result)
    }

  node_1_index    <- unique(sapply(files_names,FUN=function(filename){head(strsplit(file
name,split="_")[[1]],n=1)},USE.NAMES = FALSE))

  donor_node      <- sapply(merged.files[,"donor"], check_node, node_1_index, USE.NAMES
=FALSE)
  receiver_node   <- sapply(merged.files[,"receiver"],check_node, node_1_index, USE.NAME
S=FALSE)

  merged.files    <- cbind(merged.files, donor_node, receiver_node)

  # Exchanged in general
  all.metabolites <- setNames(aggregate(x=merged.files[,c(7,9)],by=list(merged.files$co
mpound),mean),
                              nm=c("compound","mus","smetana"))

  # Exchanged by node
  all.metabolites.by.node <- setNames(aggregate(x=merged.files[,c(7,9)],
                               by=list(merged.files$compound,merged.files$donor_node,
merged.files$receiver_node),
                               mean,),nm=c("compound","donor_node","receiver_node","mu
s","smetana"))

  # Ordered
  all.metabolites <- all.metabolites[order(all.metabolites$smetana,decreasing=TRUE),]
  all.metabolites.by.node <- all.metabolites.by.node[order(all.metabolites.by.node$smeta
na,decreasing=TRUE),]

  # Print report
  # ============
  write("The following files are empty, which means no exchange between species:\n",file
=report_name)
  write(empty,file=report_name,append=TRUE)
  write(paste0("\nThere are ",length(not.empty)," files which are not empty:"),file=repo
rt_name,append=TRUE)
  write(not.empty,file=report_name,append=TRUE)
```

```
  write("\n10 most exchanged metabolites:",file=report_name,append=TRUE)
  write.table(all.metabolites[1:10,],file=report_name,append=TRUE,quote=FALSE,sep="\t",r
ow.names=FALSE)

  write("\n10 most exchanged metabolites by donor node:",file=report_name,append=TRUE)
  write.table(all.metabolites.by.node[1:10,],file=report_name,append=TRUE,quote=FALSE,se
p="\t",row.names=FALSE)
  write("\n'first' node includes ",file=report_name,append=TRUE)
  write(node_1_index, file=report_name,append=TRUE)
  write("\nThe whole dataset is as follows:",file=report_name,append=TRUE)
  write.table(merged.files,file=report_name,append=TRUE,quote=FALSE,sep="\t",row.names=F
ALSE)
}
```

Uso: parser.R -d/-g input_file -c cores –report_name reportname

Argumentos:

- **`-g, --global`**. Carpeta que contenga informes del modo global de Smetana (e. g. ./my_results/NodeXXXXX/smetana_results/global).
- **`-d, --detailed`**. Carpeta que contenga informes del modo detallado de Smetana (e. g. ./my_results/NodeXXXXX/smetana_results/detailed).
- **`-c, --cores`**. Número de núcleos que usar en la paralelización.
- **`--report_name`**. Ruta y nombre del archivo de salida. Por defecto es `./report.txt`.

Descripción: Una vez leída la carpeta de entrada, se sigue un proceso distinto dependiendo de si es del modo global o del detallado.

Si es del modo global, primero se unen todos los archivos en un solo `data.frame`, leyéndolos con `mclapply` para mayor velocidad. Después se filtran y guardan (en `not_growing`) aquellas filas que contengan una MRO no disponible. Los nombres de las filas del `data.frame` coinciden con los de los modelos. Si la MRO es distinta de 0 en al menos un caso, el *script* continúa y obtiene el valor máximo, el medio y el mínimo de la MRO y la MIP. Finalmente se crea un informe que contiene una lista de aquellos archivos donde la MRO no estaba disponible, los valores de MRO y MIP y finalmente una lista completa de todos los resultados.

Si la carpeta dada es de resultados del modo detallado, primero se leen todos los archivos y se guardan en una lista. Esta lista se divide en dos una conteniendo los informes de Smetana vacíos y otra los que no están vacíos. Estos dos primeros pasos están paralelizados con `mclapply`.

Como los nombres predeterminados de los informes de Smetana (generados con `modelado.R`) incluyen los nombres de los dos modelos, uno de cada nodo, este *script* crea dos listas, `first` y `second`, que contienen los nombres de todos los modelos que pertenecen a cada uno de los nodos según su localización en el nombre del archivo. Esto se hace con la función `check_node`. A continuación se agrupan los metabolitos intercambiados de cada informe según el sentido en el que se intercambian, es decir, si van desde el nodo `first` hacia el `second` o viceversa (`all.metabolites.by.node`). También se crea una lista en la que se agrupan los metabolitos independientemente del sentido de intercambio (`all.metabolites`). Estas listas se ordenan de mayor a menor puntuación Smetana y se reportan en el informe final. También se incluye en el informe qué informes estaban vacíos.

Salida: Informe en formato de texto plano.

Requiere: R (3.5.0.), optparse, parallel

---

consenso.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 19 11:42:54 2020

@author: Silvia Talavera Marcos
"""

import os
import sys
import xml.etree.ElementTree as ET
import time, datetime

start = time.time()

# ================================================================================
# Leer todos los modelos
# ================================================================================
# INPUT: consenso.py input_folder output_folder outputname
# The models should not be gap-filled, as this modifies the fields
wd = sys.argv[1]
os.chdir(wd)

if len(sys.argv) > 2:
    outputdir = sys.argv[2].rstrip("/") # si el usuario da el outputdir

    if len(sys.argv) > 3:
        nodename = sys.argv[3] # si el usuario da el nombre del output
    else:
        nodename = wd.split("/")[-1]
else:
    outputdir = wd

models = [] # guardamos aquí los nombres de los archivos
for filename in os.listdir(wd):
    if ".xml" in filename:
        models.append(filename)


# ================================================================================
# Preparar el formato SBML
# ================================================================================
ET.register_namespace('', "http://www.sbml.org/sbml/level3/version1/core")
ET.register_namespace('fbc',"http://www.sbml.org/sbml/level3/version1/fbc/version2")

# ================================================================================
# Crear el "esqueleto" de nuestro modelo consenso. Usamos un modelo cualquiera.
# ================================================================================
try:
    first_model = ET.parse(models[0])
    consensus   = first_model
    cons_root   = consensus.getroot()
    cons_root[0].attrib['id'] = nodename
    cons_root[0][0][0][0].text = 'Description: This model is a consensus model from SBM
L FBC2 CarveMe output'
except:
    print("No se ha podido cargar ningún modelo.")

    raise(SystemExit(0))
```

```
# ================================================================================
# Hacer un conteo de cuántas veces aparece cada elemento en todos los
# modelos. *Evitamos cargar todos en memoria a la vez.*
# Nos quedaremos con aquellos que aparecen en el 80% o más de los modelos
# (4 de cada 5)
# ================================================================================

# Inauguramos una lista de reacciones de crecimiento (a partir de la cual obtendremos la
reacción de biomasa consenso)
growth = [first_model.getroot()[0][4][-2]]
cons_root[0][4].remove (cons_root[0][4][-2]) # y la eliminamos

# Hago una lista de diccionarios donde se hará el conteo
conteo = [{},{},{}]

# Los ID de cada elemento tienen diferente nombre según el campo
ID = ['id', 'id', '{http://www.sbml.org/sbml/level3/version1/fbc/version2}id']

# Inauguramos esta lista con el primer modelo
for n, campo in enumerate([2, 4, 6]):
    conteo[n] = {cons_root[0][campo][e].attrib[ID[n]]:1 for e in range(len(cons_root[0]
[campo]))}
    # <dicci>   <------------key------------------->
del(first_model)

# Vamos abriendo los demás modelos y contando
# Guardamos los elementos de los campos 2, 4 y 6 que no estaban ya
for m in models[1:]:
    model = ET.parse(m)
    model_root = model.getroot()
    # dejamos aparte la función de crecimiento
    growth.append(model_root[0][4][-2])
    model_root[0][4].remove(model_root[0][4][-2])

    for n, campo in enumerate([2, 4, 6]):
        for e in range(len(model_root[0][campo])):
            ident = model_root[0][campo][e].attrib[ID[n]]
            if ident not in conteo[n].keys():
                conteo[n][ident] = 1
                      #dic  #<-------key------------------------>
                cons_root[0][campo].append(model_root[0][campo][e])
            else:
                conteo[n][ident] += 1

    del(model) # !!! haciendo esto borro en memoria
    del(model_root)

# Filtro y elimino las entradas que están en menos de un 80% de modelos:
total = len(models)
for n, campo in enumerate([2,4,6]):
    removed = 0 # actualizamos el índice para evitar errores de indexación tras borrar
    for e in range(len(cons_root[0][campo])):
        ident = cons_root[0][campo][e-removed].attrib[ID[n]]
        if conteo[n][ident] < 0.80*total:
            cons_root[0][campo].remove (cons_root[0][campo][e-removed])
            removed+=1

# ================================================================================
# Por último añadimos la reacción de crecimiento que más se repite de la lista
```

```
# ================================================================================
raw_list = [tuple([tuple([str(i[n].attrib) for n in range(len(i))]) for i in item]) for
item in growth]

indexer={}
for index, i in enumerate(pure_list2):
    if i in myd.keys():
        pass
    else:
        indexer[i]=index

most_common = max(set(raw_list), key=pure_list.count) # Devuelve en formato tupla la fun
ción de crecimiento que más se repite
consensus_growth = growth[indexer[most_common]] # La tomamos en formato XML gracias a in
dexer
cons_root[0][4].append(consensus_growth)

# Guardamos el modelo
consensus.write(outputdir+"/"+nodename+"_consensus.xml", xml_declaration=True)

end = time.time()
print("Running time: ",str(datetime.timedelta(seconds = end - start)))
```

Uso: consenso.py input_folder output_folder outputname

Argumentos:

- `input_folder`. Una carpeta que contenga todos los modelos SBML-FBC2 a partir de los cuales se desee generar un consenso. Puede ser, por ejemplo, una de las carpetas generadas con `modelado.R`.
- `output_folder`. Carpeta de salida del consenso.
- `outputname`. Nombre del consenso (+ *consensus.xml*).

Descripción: Este *script* genera un modelo consenso en formato SBML-FBC2 a partir de archivos en el mismo formato. Está optimizado para modelos que no hayan sido sometidos a *gap-filling*, ya que esto modifica los campos del SBML. El formato SBML es un tipo de XML con los siguientes campos:

- root[0][0] –> notas del formato (notes)
- root[0][1] –> lista de compartimentos (listOfCompartments)
- **root[0][2] –> lista de compuestos (listOfSpecies)**
- root[0][3] –> lista de parámetros (listOfParameters)
- **root[0][4] –> lista de reacciones (listOfReactions)**
- root[0][5] –> lista de objetivos (listOfObjectives)
- **root[0][6] –> lista de productos génicos (listOfGeneProducts)**

Lo que hace este *script* es fijarse en los campos 2 (compuestos), 4 (reacciones) y 6 (productos génicos) e incluir en un archivo XML final (el consenso) todas las entradas de esos campos que estén en más de un 80% de modelos. Las reacciones del campo 4 es el que determina el crecimiento de cada modelo metabólico, mientras que el campo 6 es un campo propio del formato FBC2 que define productos génicos únicos para cada gen. Como los campos 2 y 6 dependen de las reacciones, también los tenemos en cuenta.

En primer lugar se crea un modelo XML vacío que hace de esqueleto del consenso. Este modelo está formado por todos los campos del primer modelo de la carpeta de entrada, excepto la reacción de biomasa (crecimiento) del campo 4. Paralelamente, creamos una variable (`growth`) donde se guardará, aparte, todas las reacciones de biomasa de todos los modelos con sus reactivos y coeficiente originales. Empezamos guardando la del primer modelo. También se crea una tercera variable, `conteo`, que es una

lista de diccionarios donde se apuntará cuántas veces aparece cada reactivo (campo 2), cada reacción que no sea la de crecimiento (campo 4) y cada producto génico (campo 6).

En segundo lugar, un bucle va recorriendo los demás modelos de la carpeta de entrada. A cada modelo abierto se le extrae la reacción de crecimiento y se guarda en `growth` y posteriormente se recorren sus campos 2, 4 y 6. Los campos 0 (descripción del modelo), 1, 3 y 5 no se cambian. Cada elemento `e` de cada campo (reactivo, reacción o producto génico en cada caso) se cuenta en el diccionario de `conteo`. Si no estaba ya incluido en el consenso, se incluye. Cada modelo se elimina de la memoria después de ser recorrido en el bucle.

Finalmente, un segundo bucle recorre los campos 2, 4 y 6 del consenso y elimina aquellas entradas cuyo número de apariciones en `conteo` sea menor del 80% del total de modelos. Por último, se selecciona la función de crecimiento más común entre todos los modelos y se añade al consenso final, que se guardará como *.xml*.

Salida: Modelo consenso en formato SBML-FBC2.

Requiere: Python (3.6+; en este Trabajo se usa 3.6.9).

---

consenso_EGG.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Nov 20 11:18:55 2020

@author: Silvia Talavera Marcos
"""

import os
import sys
from carveme.reconstruction.eggnog import load_eggnog_data
import datetime, time

start = time.time()

# ================================================================================
# Leer todos los modelos
# ================================================================================
# INPUT: consenso_EGG.py input_folder output_folder (outputname) (percentage)
# Only reads files that contain "annotations" in its name
# The path specified for the input_folder must be the realpath

wd = sys.argv[1].rstrip("/")

if len(sys.argv) > 2:
    outputdir = sys.argv[2].rstrip("/") # si el usuario da el outputdir

    if len(sys.argv) > 3:
        outputname = sys.argv[3].rstrip("/") # si el usuario da el nombre del output
        if len(sys.argv) > 4:
            perc = float(sys.argv[4]) # si el usuario da el porcentaje para filtrar
        else:
            perc = 0.80
    else:
        outputname = wd.split("/")[-1]


else:
```

```
    outputdir = wd

models = [] # guardamos aquí los nombres de los archivos

for filename in os.listdir(wd):
    if "annotations" in filename: # ignore seed orthologs files
        models.append(wd+"/"+filename)

if len(models) == 0:
    quit("The input file is empty.")
else:
    num_of_models = len(models)


# ================================================================================
# Crear un almacén de todas las reacciones a partir de un modelo cualquiera
# ================================================================================
all_reactions = load_eggnog_data(models[0],drop_unused_cols=False)

# Agrupamos por reacción para evitar repeticiones:
all_reactions = all_reactions.sort_values(by='score', ascending=False) \
                             .groupby('BiGG_gene', as_index=False).apply(lambda x: x.i
loc[0])
# Índice para acceder eficientemente a la fila de cada reacción:
reac2idx = {reac:idx for idx,reac in enumerate(all_reactions["BiGG_gene"])}
last_index = len(all_reactions)-1

# ================================================================================
#  Voy abriendo los demás y contando las apariciones de cada reacción
# ================================================================================

# Inicializamos el diccionario de conteo con las reacciones del primer modelo
conteo = {reac:1 for reac in all_reactions["BiGG_gene"]}
for model in models[1:]:
    # abrimos el modelo
    open_model = load_eggnog_data(model,drop_unused_cols=False)
    open_model = open_model.sort_values(by='score', ascending=False) \
                           .groupby('BiGG_gene', as_index=False).apply(lambda x: x.iloc
[0])

    for i, df_row in open_model.iterrows():
        reaction = df_row["BiGG_gene"]

        if reaction not in conteo.keys():
            conteo[reaction] = 1          # la contamos
            all_reactions = all_reactions.append(df_row,ignore_index=True) # la añadimo
s al almacén
            last_index += 1               # y anotamos su índice
            reac2idx[reaction] = last_index


        else:
            conteo[reaction] += 1         # la contamos
            # puntuación nueva ponderada (el resultado es la media para esa reacción):
            old_score = all_reactions.at[reac2idx[reaction],"score"]
            new_score = ( old_score *(conteo[reaction]-1) + df_row["score"] ) /conteo[r
eaction]
            all_reactions.at[reac2idx[reaction],"score"] = new_score # actualizar
```

```
    # cerramos el modelo (lo quitamos de memoria)
    del(open_model)


# ================================================================================
# Filtro y elimino las reacciones que están en menos de un 80% de modelos:
# ================================================================================
for r in reac2idx.keys():
    if conteo[r] < int(perc*num_of_models):
        all_reactions = all_reactions.drop(reac2idx[r],axis=0)
# ================================================================================
# Guardo el archivo de anotaciones consenso
# ================================================================================
if not os.path.exists(outputdir):
    os.mkdir(outputdir)
f = open(outputdir+"/"+outputname+".tsv","a+")

f.write("# emapper version: emapper-2.0.1 emapper DB: 2.0\n")
f.write("# consensus annotation file for "+outputname+"\n")
f.write("# time: "+str(datetime.datetime.now())+"\n")
f.write("#query_name    seed_eggNOG_ortholog    seed_ortholog_evalue    seed_ortholog_s
core best_tax_level  Preferred_name  GOs EC  KEGG_ko KEGG_Pathway    KEGG_Module KEGG_R
eaction   KEGG_rclass BRITE   KEGG_TC CAZy    BiGG_Reaction   taxonomic scope eggNOG OG
s  best eggNOG OG  COG Functional cat. eggNOG free text desc.\n")
all_reactions.to_csv(f,sep=" ",header=False, index=False)

f.close()

end = time.time()
print("Running time: ",end - start)
```

Uso: consenso_EGG.py input_folder output_folder (outputname) (percentage)

Argumentos:

- **`input_folder`**. Una carpeta que contenga todos los archivos de anotaciones de eggNOG-mapper a partir de los cuales se desee generar un consenso. Los archivos de anotaciones deben contener *annotations* en el nombre.
- **`output_folder`**. Carpeta de salida del consenso.
- **`outputname`**. Nombre del consenso (+ *.tsv*).
- **`percentage`**. Porcentaje de archivos en los que debe estar presente una anotación funcional para ser incluida en el consenso final. Por defecto es 80%. Debe indicarse con el formato “0.80”.

Descripción: Este *script* genera un archivo consenso de anotaciones funcionales a partir de una carpeta dada que contenga múltiples archivos de anotaciones funcionales, obtenidos con eggNOG-mapper. El formato de estos archivos es variable según las opciones indicadas a eggNOG-mapper; el formato compatible en este caso es aquel que se genera con *annotate.R* y que CarveMe es capaz de leer con su opción `--egg` para generar modelos.

En primer lugar, se lee la carpeta de entrada. Si hay al menos un modelo con *annotations* en el título, el *script* continúa. En segundo lugar, se inicializa con la función `load_eggnog_data` u n `DataFrame` de `pandas`, llamado `all_reactions`, a partir del primer modelo de la lista. También se inicializa un diccionario de conteo de reacciones.

`all_reactions` incluirá todas las reacciones diferentes que aparezcan en los modelos de la carpeta de entrada y se llena en un bucle que recorre los demás modelos. Se abre cada modelo, se seleccionan sus

reacciones únicas y se contabilizan en `conteo`. Como cada reacción tiene una puntuación asignada, utilizada por CarveMe para hacer modelos, cada vez que se repite una reacción se actualiza su puntuación haciendo una media ponderada. También se van anotando los índices de cada reacción para facilitar los pasos posteriores. Tras cada iteración, se elimina de memoria el último modelo abierto.

Por último, se aplica el filtro: se eliminan de `all_reactions` aquellas reacciones que no aparezcan en al menos el porcentaje especificado de archivos de anotaciones originales. La lista filtrada se guarda como `.tsv`. Posteriormente esta lista de anotaciones consensuada podrá ser utilizada por CarveMe para generar un modelo consenso.

Salida: Archivo de anotaciones funcionales *.tsv*.

Requiere: Python (3.6+; en este Trabajo se usa 3.6.9), CarveMe (1.4.0+, incluye pandas).